Computação & Objetos

Parte 2

Livro do Estudante - 1° Ano

Projeto Computação na Vida

Francisco Tito Silva Santos Pereira
Roberto Almeida Bittencourt

Versão 1.0

Autores: Francisco Tito Silva Santos Pereira e Roberto Almeida Bittencourt

Neste livro utilizamos imagens de comandos e da linguagem de programação Python e do framework PPlay para desenvolvimento de jogos. O PPlay foi desenvolvido no Instituto de Computação da UFF para ser usado como ferramenta de auxílio no ensino de programação aos alunos ingressantes. Mais detalhes, no seu link oficial: http://www2.ic.uff.br/pplay/
Algumas imagens utilizadas estão disponíveis sob a licença Creative Commons Attribution-ShareAlike 2.0 International (CC BY-SA 2.0). A imagem animada do porco, utilizada nas aulas 5 e 6 foi criada por Jeremie Kent Leyva (com o usuário Redbird25) e disponibilizado em um fórum para fãs do jogo Angry Birds: https://angrybirds.fandom.com/wiki/User:Redbird25

Os códigos-fonte de cada jogo estão disponíveis na pasta http://bit.ly/CodigosJogosPPlay .

Sumário

Resultados de Aprendizagem:

Pensamento Computacional (PC)

PC01. Usar passos básicos na solução de problemas algorítmicos para projetar soluções (por exemplo, declaração e exploração de problemas, exemplos de instâncias, design, implementação de uma solução, teste, avaliação)

PC06. Descrever e analisar uma sequência de instruções que estão sendo seguidas (por exemplo, descrever o comportamento de um personagem em um videogame conforme orientado por regras e algoritmos).

PC07. Representar dados de várias formas, incluindo texto, sons, imagens e números.

PC11. Analisar o grau em que um modelo de computador representa com precisão o mundo real.

PC12. Usar abstração para decompor um problema em subproblemas.

PC15. Fornecer exemplos de interdisciplinaridade em aplicações do pensamento computacional

Colaboração (C)

C6. Criar, desenvolver, publicar e apresentar, de forma colaborativa, produtos (por exemplo, vídeos, podcasts, websites) usando recursos de tecnologia que demonstram e comunicam conceitos do currículo.

C7. Colaborar com colegas, especialistas e outras pessoas usando práticas colaborativas, como programação em pares, trabalho em equipes de projeto e participação em atividades de aprendizado ativo em grupo.

Prática de Computação e programação (PCP)

PCP04. Demonstrar compreensão de algoritmos e sua aplicação prática.

PCP05. Implementar soluções de problemas usando uma linguagem de programação, incluindo: comportamento de looping, instruções condicionais, lógica, expressões, variáveis e funções.

AULA 1 - Criando o jogo Pong

1. Sumário

Nesta aula você irá aprender os conceitos básicos da criação de jogos digitais utilizando a linguagem de programação **Python.** Além de compreender os conceitos iniciais da programação orientada a objetos, que é uma nova forma de se programar e que será bastante útil na criação dos jogos.

TÓPICOS RELEVANTES

Programação Orientada a Objetos

Vamos conhecer um pouco como funciona a Programação Orientada a Objetos, esta que estaremos utilizando durante o nosso livro.

Vejamos o seguinte exemplo:

*Surge a oportunidade para criarmos um jogo de espaçonaves. Neste jogo, teremos a nossa nave e para vencermos, precisamos destruir as naves inimigas e o chefão que irá aparecer no final do jogo. A nossa nave possui uma **cor azul** e os seus **tiros** possuem a **cor branca**. Já os inimigos podem possuir vários tipos de cores, vermelho, amarelo, branco, entre outros (só não pode ser azul, pois a nossa nave já é azul). Por fim, o chefão também possuirá um esquema de cores que irá compor a sua carcaça e ao contrário das outras naves, ele irá soltar dois tiros por vez.*

A partir dessa descrição que nos foi passada, podemos perceber alguns detalhes que são extremamente importantes para a Programação Orientada a Objetos, vamos destacar alguns pontos que são importantes:

- Uma nave irá possuir uma cor (a depender da nave ela pode possuir cores diferentes)
- Uma nave pode atirar (Se for o chefão, ele irá soltar dois tiros por vez)
- Uma nave pode se movimentar pelo espaço

A Programação Orientada a Objetos tem como característica principal aproximar os detalhes da programação com o mundo real.

Podemos chamar os diversos tipos de naves que aparecem na nossa tela de **objetos**.
Percebemos que esses **objetos** possuem suas características que os diferenciam no mundo. Por exemplo, a nossa nave possui uma cor azul e, seus tiros, a cor branca - diferentemente de um inimigo que possui a cor amarela e o tiro de cor vermelha, e até de outro inimigo que possui a cor laranja.

Podemos pensar também que podemos ter o controle da nossa nave, fazendo-a se movimentar - geralmente, utilizamos as setas para esse controle. Pensando na nossa nave com um objeto único, esse controle das setas só irá servir para a nossa nave, pois os outros objetos se comportam de maneira diferente e um objeto não interfere no outro.

Quais são todas as **características** que essas naves possuem em comum?

➔ Cor principal
➔ Cor do tiro
➔ Quantidade de tiros disponíveis por vez
➔ Posição no eixo horizontal

➔ Posição no eixo vertical

Quais são todos os **comportamentos** que essas naves possuem em comum?

➔ Movimentar
➔ Atirar

Ao perceber e definir essas características e comportamentos em comum de um objeto, podemos introduzir um novo conceito: **classe**.

Uma **classe** funciona como um **molde** para um objeto. Por exemplo:

➔ Vimos que todas as características de uma nave são a sua cor principal, cor do tiro e quantidade de tiros disponíveis por vez.

Com a classe, nós iremos somente definir quais são essas características, pois lembre-se, ela funciona como um **molde** para o objeto. Ao criarmos um objeto a partir desta classe, o objeto vai assumir valores para as suas características. Por exemplo, a cor principal e a cor do tiro podem ser **vermelhos**, e a quantidade de tiros disponíveis por vez pode ser **1**.

Com a programação orientada a objetos, nós chamamos as características de um objeto de **atributo**, que são basicamente variáveis pertencentes ao objeto e que vão assumir valores específicos. E os comportamentos de um objeto, nós chamamos de **métodos**, que são funções pertencentes ao objeto e que vão descrever como cada objeto se comporta. Os métodos podem ser iguais para todos os objetos de uma mesma classe (por exemplo, movimentar pode ser similar para todas as naves) ou serem diferentes a depender do objeto. Por exemplo, quando o chefão atira, são disparados dois tiros por vez, enquanto que uma nave comum dispara apenas um tiro por vez.

A partir desse entendimento inicial, podemos criar uma **classe** e seus **objetos** com a programação em **Python**, como no exemplo a seguir:

```
    class Nave:

 1    def __init__(self):
 2      self.cor = ""
 3      self.cor_do_tiro = ""
        self.qtd_tiros = 0
 4
 5    def movimentar(self):
 6        # Aqui haverá um código que irá fazer o objeto do tipo Nave se
 7  #movimentar
 8
 9    def atirar(self):
10        # Aqui haverá um código que irá fazer o objeto do tipo Nave
11  #atirar
12
13  # O código acima define uma classe Nave.
14  # Abaixo, você pode ver um programa que usa esta definição para criar
15  #objetos do tipo Nave.
16  nave1 = Nave()
17  nave1.cor = "vermelho"
18  nave1.cor_do_tiro = 'azul'
19  nave1.qtd_tiros = 1
20  nave1.atirar()
21
22  nave2 = Nave()
23  nave2.cor = "amarelo"
24  nave2.cor_do_tiro = "vermelho"
25  nave2.qtd_tiros = 3
26  nave2.atirar()
27
```

Na linha 16, estamos **instanciando** um objeto, ou seja, estamos criando um objeto a partir de um molde (classe). Podemos acessar os seus atributos a partir do ponto (.), ou seja, quando temos na linha 17:

```
nave1.cor = "vermelho"
```

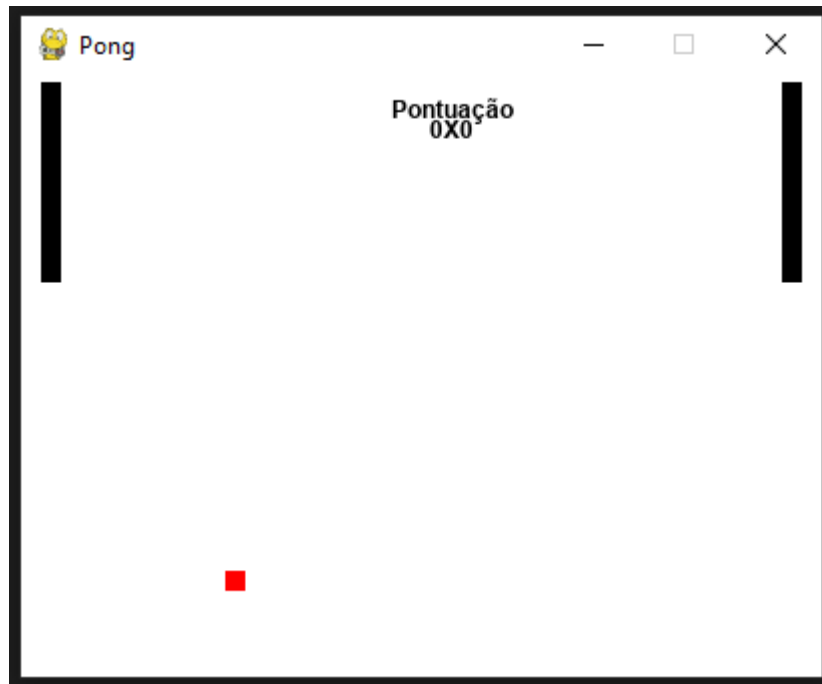
Estamos acessando o atributo do nosso objeto que chamamos de `nave1`, e estamos atribuindo um valor, que neste caso é vermelho.
Faremos o mesmo com a cor do seu tiro e a quantidade tiros disponíveis, apresentados na linha 18 e 19, respectivamente.

Por fim, na linha 20, chamamos o método `atirar`. Também utilizamos o ponto (.) para acessar os métodos de um objeto. Como foi definido anteriormente dentro da classe, o método atirar,

quando for chamado, irá mostrar uma mensagem na tela, que a depender dos valores que o objeto que a chamou possui, irá ser mostrada de maneira diferente.

Na linha 22, criamos um outro objeto a partir da classe Nave, esta terá outras características que serão diferentes do objeto que criamos anteriormente. Ela terá uma cor amarela, a cor da bala será vermelha e ela poderá atirar 3 balas por vez.

Neste tópico importante aprendemos um pouco sobre a **Programação Orientada a Objetos**. Nela teremos **classes**, as quais serão moldes para **objetos**. Um **objeto** será **único** no mundo e ele não poderá interferir com outro já criado. Um **objeto** para existir terá que possuir **atributos** que são as suas **características** e **métodos** que são os seus **comportamentos**. A depender do objeto, ele pode se **comportar** de **maneira diferente** no mundo.

FOLHA DE ATIVIDADES - Criando o jogo Pong

O Pong é um jogo clássico, ele foi o primeiro jogo lucrativo da história e possui uma grande importância na indústria dos jogos.

Ele foi feito para dois jogadores. Um irá controlar a barra direita e o outro a esquerda. A bola irá aparecer no centro da tela e aleatoriamente irá escolher uma posição para se movimentar. Caso a bola colida no canto direito da tela, o jogador da esquerda ganha um ponto. Porém, se a bola colidir no canto esquerdo da tela, o jogador da direita ganha um ponto
Os jogadores irão movimentar a barra para que a bola não colida com seu canto da tela.
Ganha o jogador que fizer 3 pontos primeiro.

Para a implementação do jogo, iremos utilizar o **PPlay** que utiliza o **Pygame**.

Inicialmente, será necessário baixar os arquivos importantes para a criação do jogo. Acesse o seguinte link: http://bit.ly/ArquivoPong

Em seguida, nesta mesma pasta, no diretório principal, crie um arquivo chamado **pong.py**. Nele iremos implementar o nosso código.
Como foi dito anteriormente, utilizaremos o **PPlay** para criação de jogos. Ele já possui várias ferramentas prontas que irão nos auxiliar nessa jornada, sendo assim, teremos que realizar uma importação dessas ferramentas que irão aparecer através das **classes**.

Passo 01 - Criando a tela do jogo

Na primeira linha do seu código, escreva o seguinte:

```
from PPlay.window import *
```

from - Palavra reservada que indica de onde iremos importar os nossos arquivos
PPlay - Pasta que está no seu diretório
.window - Nome do arquivo que está dentro da pasta PPlay
import - Palavra reservada que irá fazer a importação dos arquivos
***** - Significa que iremos importar tudo que está dentro do arquivo window

Feito isso, importamos uma classe importante do PPlay que irá trabalhar especificamente com a tela do jogo: Window. Assim, iremos criar um objeto do tipo Window logo após a sua importação:

```
janela = Window(400,300)
```

Acabamos de criar um objeto do tipo Window, que estará armazenado na variável janela. Neste caso, quando criamos o objeto do tipo Window, temos que passar como parâmetro duas variáveis: as dimensões da tela. Neste caso, utilizaremos uma tela com 400 pixels de largura e 300 pixels de altura.

Agora para desenhar a janela na tela do seu computador, iremos chamar o método `update()` que está acessível para o objeto, basta escrever:

```
janela.update()
```

E rodar o seu código!

Se você viu rapidamente uma tela preta aparecendo na tela, não se preocupe pois isso era o esperado!

O nosso código está sendo executado linearmente. Após você escrever `janela.update()`, o interpretador do python leu esse código, executou, viu que não tinha mais nenhum código abaixo deste e finalizou o programa.

Para evitar isso, utilizaremos um conceito que chamamos de **game loop.** Pegando a tradução literal deste termo, temos um **ciclo de jogo.** Na programação, quando falamos de ciclos, podemos pensar rapidamente nos **loops** ou nas **estruturas de repetição.** Sendo assim,
teremos uma **repetição** dentro do nosso código, e todo o nosso jogo irá rodar a partir dessa repetição.

O **game loop** é um conceito bastante importante para a programação de jogos. Basicamente ele serve como uma estrutura de controle pois com ele, o jogo irá executar uma série de ações, essas ações podem ser: pegar as entradas do usuário (pode ser o ponteiro do mouse, um botão do teclado pressionado), a partir desta entrada do usuário modificar o estado do jogo (modificar a posição de um personagem ou até modificar o cenário que está sendo mostrado) e por fim, desenhar o jogo com as modificações realizadas.

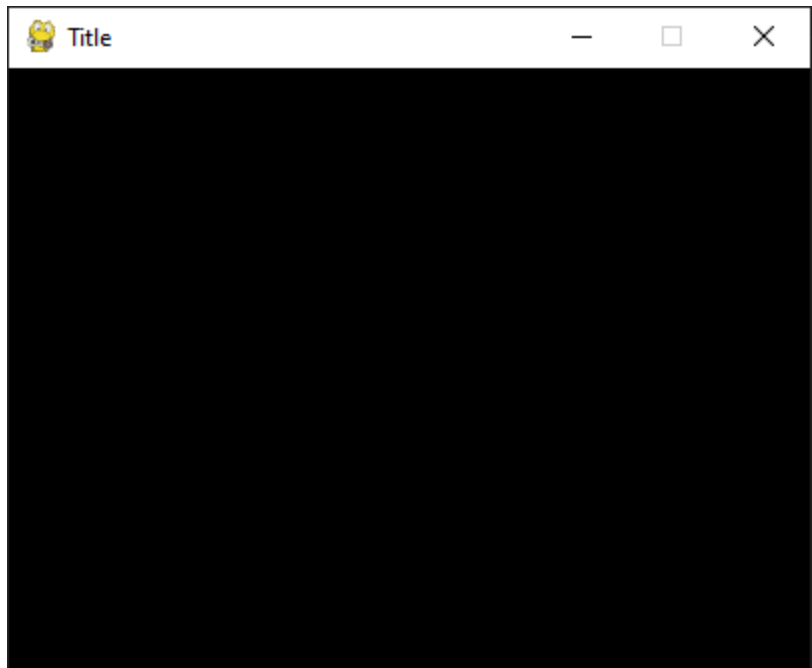
Sendo assim, os três principais pontos que acontecem durante o **game loop** são:

- Obter a entrada do usuário
- Modificar o estado do jogo
- Desenhar o jogo

É importante ressaltar que as importações de classes e instanciações de objetos deverão ser feitas **antes** do **game loop**, pois como estamos lidando com uma estrutura de repetição, caso instanciamos um objeto dentro dele, este sempre será criado e suas informações serão perdidas ao final de cada loop.

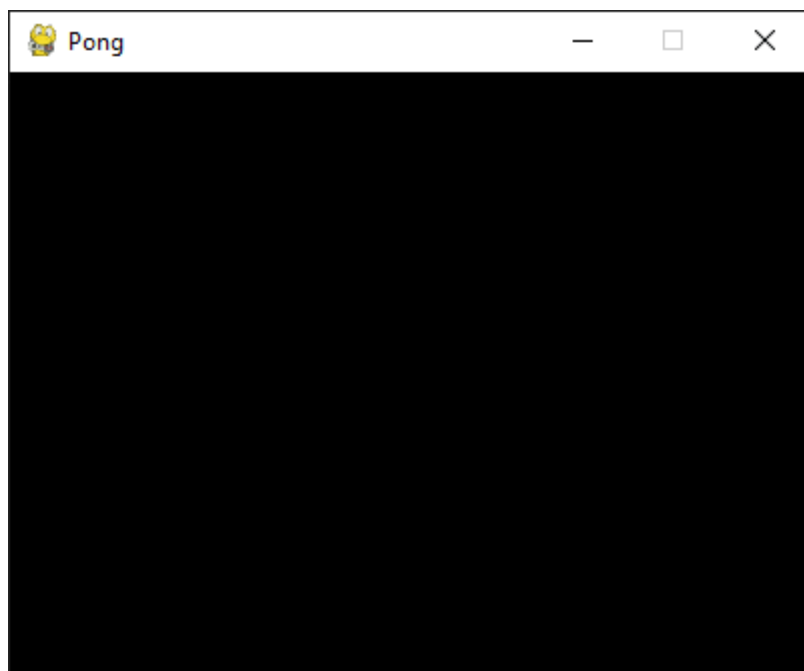
Sendo assim, teremos o nosso código desse jeito:

```
# Inicializar os dados e criar a janela do jogo
from PPlay.window import *
janela = Window(400,300)

# game loop abaixo
while(True):
      janela.update()
```

Sendo assim, agora temos a nossa janela aparecendo sem interrupções:

Title

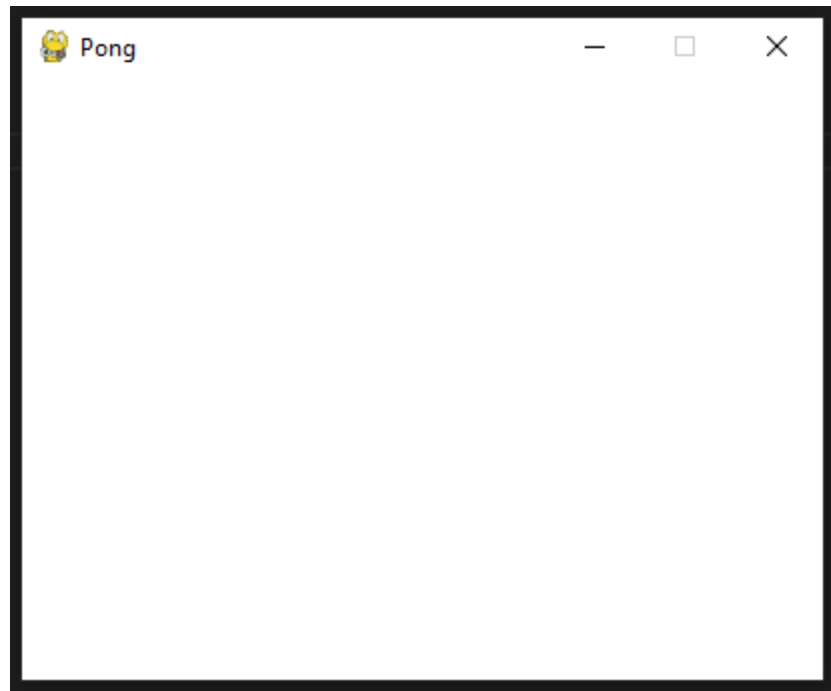
Passo 02 - Modificando a tela do jogo.

Podemos modificar o título que aparece no canto superior. Basta antes do nosso game loop, chamar o método set_title com o objeto janela, e passar como parâmetro o nome que você deseja que apareça:

```
janela.set_title("Pong")
```

Se essa tela estiver aparecendo dessa forma, tudo está conforme o planejado.
Podemos também, trocar a cor da nossa tela, pois a **classe Window** possui o método **`set_background_color()`** que irá receber três parâmetros : no primeiro valor o valor **R** (vermelho), o segundo **G** (verde) e no terceiro **B** (azul).
Para as alterações com a cor de fundo da janela funcionarem corretamente, é necessário chamar esse método **dentro** do **game loop** para que sempre que a tela for atualizada com as novas informações, a sua cor continuar sendo a mesma:

```
janela.set_background_color((255,255,255))
```

Desta forma, a sua tela irá ficar branca:

Pong

Passo 03 - Adicionando as barras do jogo.

Seguindo adiante, precisamos colocar na nossa tela as barras que irão se movimentar conforme o input do usuário. Para isso, o PPlay apresenta a classe **Sprite**. Um sprite é elemento gráfico bidimensional que pode ser controlado pelo usuário através de interações com o teclado ou mouse. É necessário associar uma imagem ao sprite para que ele apareça na tela e chame atenção do usuário para que ele possa o controlar.

Importamos a classe Sprite da seguinte forma:

```
from PPlay.sprite import *
```

Sendo assim, a classe **Sprite** foi importada com sucesso, e podemos instanciar seus objetos.
Para instanciar um objeto do tipo **Sprite**, precisamos passar a localização da imagem. Esta imagem é a que aparecerá na tela. Neste caso, a imagem da barra está presente dentro da pasta assets e com o nome de barra.png. Logo:

```
barra_esquerda = Sprite("./assets/barra.png")
barra_direita = Sprite("./assets/barra.png")
```

Acabamos de instanciar dois objetos, um será responsável pela barra que ficará no canto esquerdo da tela e outro na direita. Precisamos agora definir quais posições as barras irão ocupar inicialmente. A classe **Sprite** tem o método `set_position(x,y)` que recebe como parâmetro as posições x e y que você deseja que o seu objeto fique na tela.

Após as criação dos objetos, lembrando que será **antes** do **game loop**, podemos chamar este método para modificar as suas posições iniciais:

```
barra_esquerda.set_position(10,3)
barra_direita.set_position(380,3)
```

Se você rodar o seu programa, você irá verificar que mesmo com as importações, as barras não aparecem na tela. Isso acontece porque até agora, dentro do **game loop** a gente não chamou a função que desenha as nossas barras.

Sendo assim, **antes** de `janela.update()` iremos escrever:

```
barra_esquerda.draw()
barra_direita.draw()
```

E teremos:

AULA 2 - Continuando a criação do jogo Pong

1. Sumário

Nesta aula você irá continuar na construção do jogo Pong que você começou na aula passada. Você irá relembrar alguns conceitos dados anteriormente e finalizará o seu jogo!

TÓPICOS RELEVANTES

Uma explicação sobre Quadros por segundo

Você já deve ter ouvido falar sobre frames per second ou até frames por segundo ou até mesmo quadros por segundo, especialmente se você se interessa por games. Basicamente essa expressão está se referindo a uma sequência de imagens fixas que, quando são reproduzidas em sequência em uma certa velocidade, temos uma simulação de movimento. Esta técnica é bastante utilizada na indústria de jogos e de animações.

Você pode testar esse exemplo com o seu caderno. Faça um desenho de um boneco na ponta de uma folha do seu caderno. Na outra folha replique este mesmo desenho, porém levante um pouco o braço desse boneco. Vá levantando aos poucos este braço, até que no final o braço do boneco vai estar totalmente levantado.

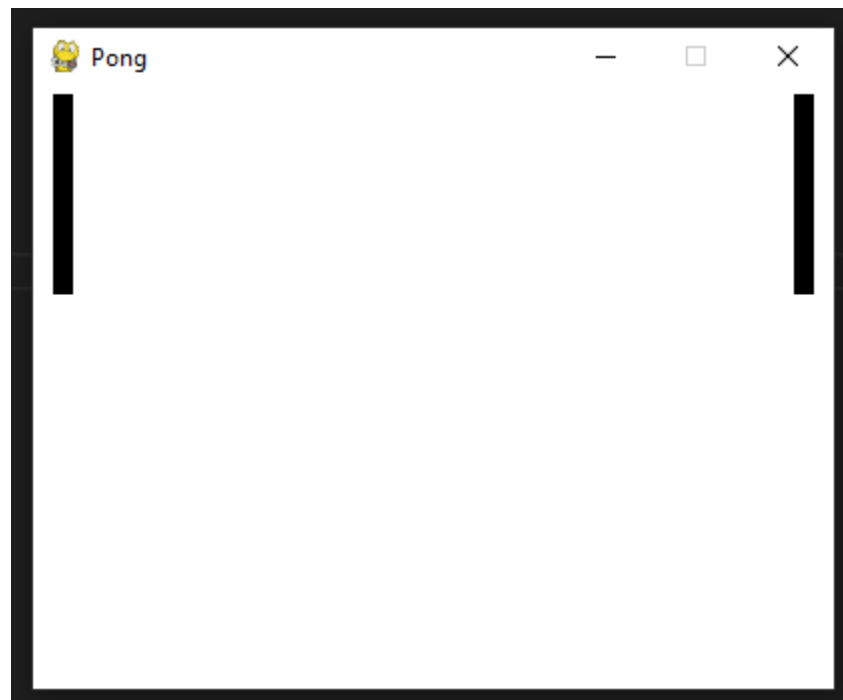
Você deve ter gastado umas 10 ou 15 páginas para poder fazer isso, não se assuste, mesmo para uma simples animação várias páginas são gastas para isso, hoje com a tecnologia e a computação gráfica, não é mais necessário que os desenhistas gastem muitas folhas de papel para fazer o desenho. Mas imaginem antigamente, quando não existia toda essa tecnologia, várias folhas de papel eram utilizadas...

Feito o desenho, você pode simular a animação ao passar rapidamente as páginas, você verá o braço do boneco indo para cima ou indo para baixo.

Quadros por segundo está se referindo a quantos quadros (imagens) aparecem na tela a cada segundo. Quanto maior este número, geralmente 60 FPS, temos uma animação mais limpa, caso contrário teremos uma animação mais brusca.

FOLHA DE ATIVIDADES - Continuando o jogo Pong

Na aula passada começamos a construção do jogo Pong, ele está sendo feito com o PPlay que utiliza a linguagem de programação Python.

Até o momento temos o seguinte:

Conseguimos colocar as duas barras no jogo.

Passo 01 - Movimentando as barras do jogo

Lidaremos agora com a movimentação do teclado, esta a qual irá movimentar as barras.
Por convenção, iremos assumir que a barra esquerda irá para baixo quando a tecla **S** for pressionada e para cima quando **W** for pressionado.
Já na barra direita temos os botões que possuem a seta para cima e para baixo.

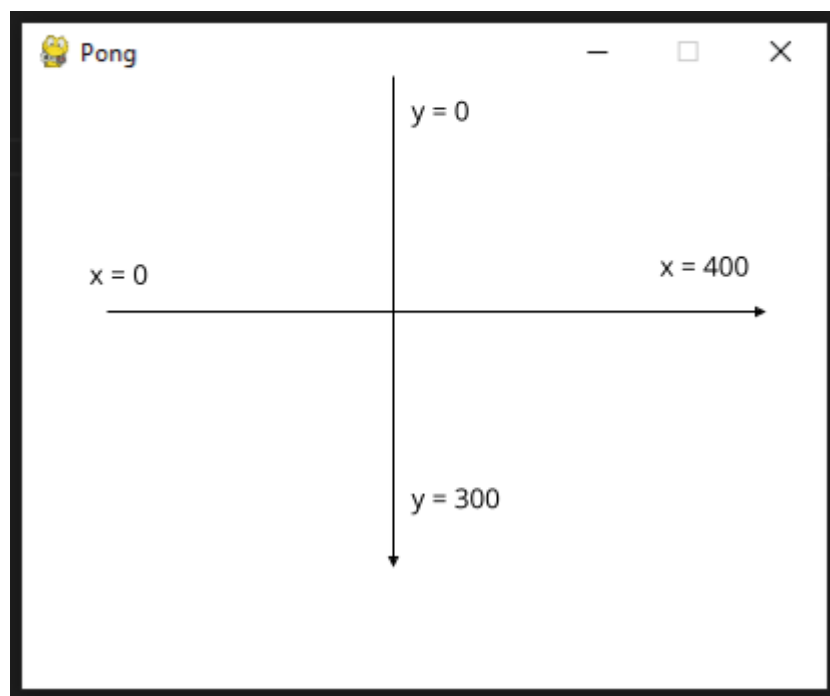
Para realizarmos o controle do teclado, iremos importar uma outra classe presente no **PPlay**: keyboard:

```
from PPlay.keyboard import *
```

E podemos instanciar o seu objeto:

```
teclado = Keyboard()
```

Ao instanciar o objeto do tipo Keyboard, não precisamos passar nada como parâmetro.

Antes de implementarmos a funcionalidade de movimentação das barras laterais, é importante saber como funciona a tela no **PPlay** em relação às coordenadas x e y:

Percebe-se que a coordenada x cresce da esquerda para a direita. No nosso caso, como temos 400px de largura, o valor mínimo de x vai ser 0px (no canto extremo esquerdo da tela) e o seu valor máximo vai ser 400px (no canto extremo direito da tela).

Já a coordenada y cresce de cima para baixo. No nosso caso, temos 300px de altura. O valor mínimo de y vai ser 0px (no canto extremo superior da tela) e o seu valor máximo vai ser 300px (no canto extremo inferior da tela).

Após a importação da classe **Keyboard**, poderemos realizar a verificação da tecla pressionada do teclado.

Iremos inicialmente realizar a movimentação no teclado esquerdo que irá utilizar as teclas **S** e **D** do teclado. Posteriormente iremos somente replicar o código e utilizar as teclas de seta para cima e para baixo...

Se queremos que uma barra se mova para baixo, será necessário aumentar a sua coordenada y. Caso queremos que ela se mova para cima, diminuímos a sua coordenada y.

Sendo assim, utilizaremos a estrutura condicional **if** juntamente com o método `key_pressed(“NOME_DA_TECLA”)` que está presente no objeto teclado. Veja a seguir:

```
if teclado.key_pressed("S"):
    barra_esquerda.move_y(60)
if teclado.key_pressed("W"):
    barra_esquerda.move_y(-60)
```

Lembrando que, esse código estará dentro do game loop, pois ele está recebendo o input do usuário.

Vamos interpretar o código:

- Caso o nosso objeto de teclado perceba que o usuário pressionou a tecla **S**, a barra direita irá modificar o seu valor de y por 60px. O que isso significa: se antes a coordenada y da barra era 3px, ao chamar o método `move_y(60)` o seu novo valor vai ser 63px. Se o usuário continuar pressionando a tecla **S**, será 63px + 60px = 123px. Ou seja, quando estamos aumentando o seu valor atual por um novo, simplesmente pegamos o seu valor atual e somamos com o novo valor.
- Já no caso de o usuário pressionar a tecla **W**, iremos diminuir os pixels da sua coordenada y para que o objeto possa ir para cima. Sendo assim, chamamos o método `move_y(-60)` para que possamos diminuir 60px da sua coordenada. Ou seja, se antes tinha 123px, agora irá ter 63px...

Vamos rodar o nosso jogo e verificar o que acontece.

Você deve ter rodado o jogo e viu alguns problemas:

1. A barra está se movimentando rápido demais
2. A barra está saindo da visão do jogador

Vamos resolver por partes.

Passo 02 - Movimentando a barra mais lentamente

Como foi dito anteriormente, temos que todo o nosso jogo roda através do **game loop** que é basicamente um laço de repetição que é infinito `(while True)`. Cada loop completo é chamado de 1 quadro, porém a depender da velocidade de um computador, nosso código irá rodar mais rápido ou não, ou seja, teremos mais ou menos quadros por segundo.

Para solucionar isso, o **PPlay** possui implementado um controle de quadros por segundo (QPS ou em inglês *Frames per second - FPS*), garantindo que um jogo possa rodar na mesma velocidade em máquinas diferentes.
Em algumas máquinas, a taxa de quadros por segundo pode ter um valor. Porém, em um computador mais veloz essa taxa de quadros por segundo pode ter um valor maior. Sendo assim, a barra pode se movimentar mais lentamente em um computador, porém em outro ele se movimenta muito mais rápido.
O que queremos é basicamente definir que esse movimento seja único independente do computador em que esse jogo esteja sendo rodado. Essa solução vem com a classe **Window** na qual temos acesso aos seus atributos e métodos a partir do **objeto janela**.
Chamaremos o método `delta_time()` que é basicamente um intervalo de tempo entre um quadro e outro. Ele irá nos retornar o intervalo entre quadros do computador de onde o jogo está sendo executado.

Ele vai ajudar a solucionar o problema da barra se movimentando rapidamente, para isso, é necessário multiplicar o valor retornado pela função `delta_time()` com o valor que queremos que a barra se movimente (neste caso 60). Logo, ao fazer uma reformulação do código, teremos o seguinte:

```
if teclado.key_pressed("S"):
      barra_esquerda.move_y(60*janela.delta_time())
if teclado.key_pressed("D"):
      barra_esquerda.move_y(-60*janela.delta_time())
```

Ao fazer isso, temos um movimento das barras mais suave.

Você pode replicar este código agora para movimentar a outra barra. Em vez de utilizar o S para subir, você usa o “UP”, e em vez de D, você utiliza o “DOWN”. (Lembre-se de utilizar o **objeto** barra_direita).

Passo 03 - Fazendo com que a barra não ultrapasse os limites do quadro

Agora vamos fazer com que a barra não ultrapasse os limites do quadrado visível. Esses limites foram definidos na imagem mostrada anteriormente, o limite inferior é 0 e o superior é 300. Sendo assim, para impedir que a barra suma da tela, enquanto ele está subindo, temos que permitir a movimentação somente quando a sua coordenada y é maior que 0.

Para o movimento contrário, teremos que levar em consideração o tamanho da barra, ele possui 100px de largura, logo em vez de fazermos a comparação com 300px (tamanho total da tela, teremos que verificar se a posição y da barra é menor que 200px:

```
if teclado.key_pressed("S") and barra_esquerda.y < 200:
    barra_esquerda.move_y(60*janela.delta_time())
if teclado.key_pressed("W") and barra_esquerda.y > 0:
    barra_esquerda.move_y(-60*janela.delta_time())
```

Sendo assim, ao fazer isso, teremos a barra se movimentando não tão rápido como antes, e não ultrapassando os limites do quadro.

Passo 04 - Adicionando a bola no jogo

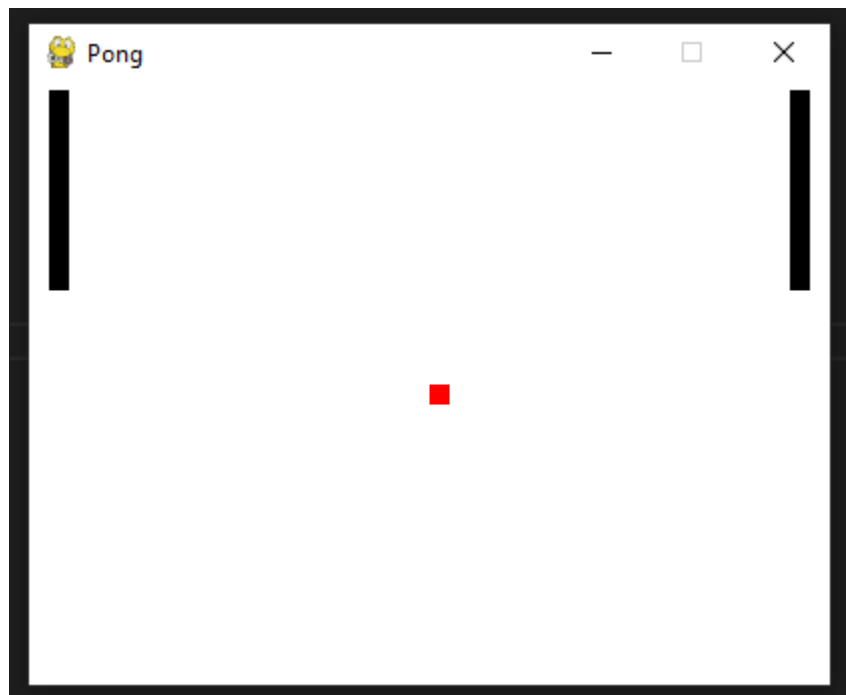
Vamos adicionar agora a bola que estará se movimentando no nosso jogo. Já temos a importação do **Sprite**, sendo assim iremos somente instanciar um novo objeto:

```
bola = Sprite("./assets/bola.png")
bola.setposition(200,150)
```

Com a primeira linha temos a instanciação do objeto e em seguida, estamos modificando a sua posição para o centro da tela.

Agora lá no nosso **game loop**, vamos desenhar a bola com o comando:

```
bola.draw()
```

Você deve ter o seguinte:

Passo 05 - Movimentando a bola

Vamos agora fazer a bola se movimentar. Para isso, vamos criar algumas variáveis auxiliares que irão armazenar o seu deslocamento x e y.

Para realizarmos a movimentação da bola, vamos criar duas variáveis que irão ser responsáveis por armazenar os valores do deslocamento da bola. Uma será responsável pelo deslocamento em x e o outro pelo deslocamento em y. Inicialmente esses valores serão 0:
(antes do **game loop** teremos o seguinte)

```
bola_deslocamento_x = 0
bola_deslocamento_y = 0
```

Após isso, iremos definir aleatoriamente para onde a bola irá se movimentar.
Para isso, vamos importar a função random, que irá gerar um número aleatório:

```
import random
```

Agora, feito a importação, e antes do **game loop**, faremos o seguinte:

```
direcao_bola = random.randint(1,4)
```

Ao fazermos random.randint(1,4) estamos chamando a função `randint()` que está presente em **random** (o qual importamos anteriormente). Vamos passar dois valores, um é o menor número que queremos que seja sorteado, no caso 1, e o segundo é o maior número que queremos que seja sorteado. Logo, estaremos sorteando um número entre 1 e 4.

A partir deste número que foi sorteado, podemos definir qual é a direção da bola, vamos pensar o seguinte:

- Caso o número sorteado for 1, queremos que a bola se movimente em x para a direita e em y para baixo
- Caso o número sorteado for 2, queremos que a bola se movimente em x para a direita e em y para cima.
- Caso o número sorteado for 3, queremos que a bola se movimente em x para a esquerda e em y para cima.

- Caso o número sorteado for 4, queremos que a bola se movimente em x para a esquerda e em y para cima.

Transformando esta ideia em código, temos o seguinte:

```
if direcao_bola==1:
    bola_deslocamento_x = 0.06
    bola_deslocamento_y = 0.06
elif direcao_bola==2:
    bola_deslocamento_x = 0.06
    bola_deslocamento_y = -0.06
elif direcao_bola==3:
    bola_deslocamento_x = -0.06
    bola_deslocamento_y = 0.06
elif direcao_bola==4:
    bola_deslocamento_x = -0.06
    bola_deslocamento_y = -0.06
```

Agora, dentro do **game loop**, vamos chamar a função `move_x()` e `move_y()` com o objeto bola e iremos passar os valores que foram definidos anteriormente e armazenamos dentro das variáveis `bola_deslocamento_x` e `bola_deslocamento_y`:

```
bola.move_x(bola_deslocamento_x)
bola.move_y(bola_deslocamento_y)
```

Se você rodar o programa, vai ver que a bola não está se movimentando. Isso porque, precisamos chamar a função a função que **desenha** o objeto na tela (dentro do **game loop** porém bem no final do código, lembre-se a última coisa que o **game loop** faz é desenhar os objetos na tela):

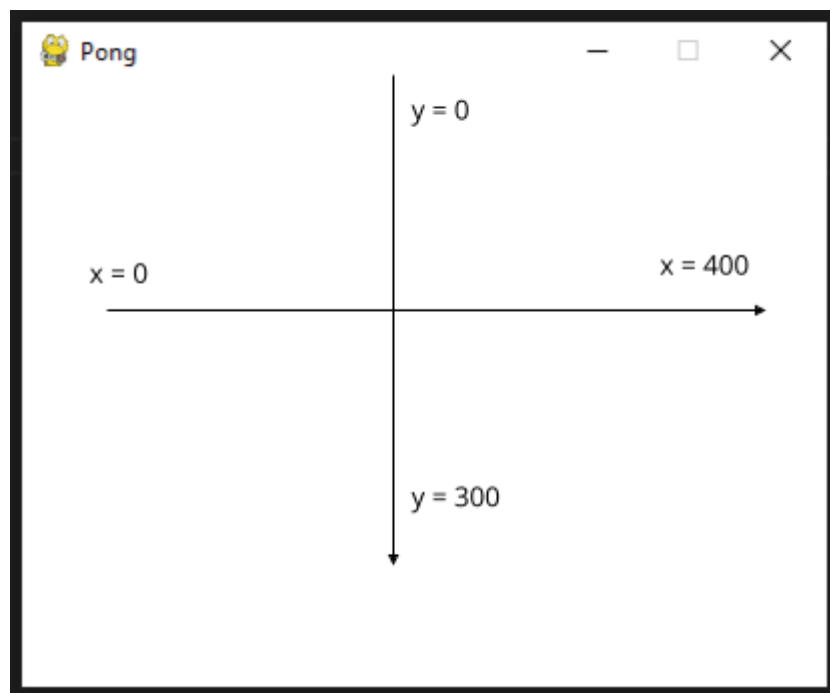
```
bola.draw()
```

Passo 06 - Ricocheteando a bola

Bom, até o momento a bola está se movimentando, porém quando ela bate na ponta ela continua avançando e sem previsão de volta.

Para isso, vamos estar verificando as coordenadas x e y do nosso objeto bola. O que iremos fazer, estará dentro do **game loop**, pois essa verificação sempre tem que ser feita a cada movimento da bola.
Vamos pensar:

- Se a coordenada y da bola for menor que 0px, significa que a bola ultrapassou o limite superior da nossa tela. Sendo assim, temos que fazer a bola descer.
- Se a coordenada y da bola for maior que 290px, significa que a bola ultrapassou o limite inferior da nossa tela. Sendo assim, temos que fazer a bola subir. (Estamos considerando que a imagem da bola possui 10px de largura e de altura, sendo assim estamos descontando os 10px dos 300px totais da altura do limite inferior).
- Se a coordenada x da bola for maior que 400px, significa que a bola ultrapassou o limite da direita da tela. Sendo assim, temos que fazer a bola ir para a esquerda.
- Se a coordenada x da bola for menor que 0px, significa que a bola ultrapassou o limite da esquerda da tela. Sendo assim, temos que fazer a bola ir para a direita.

Caso essa explicação tenha sido um pouco confusa, tente se lembrar dessa imagem:

Convertendo o pensamento em código, teremos:

```
if(bola.y >= 290):
        bola_deslocamento_y = -60*janela.delta_time()

if(bola.y < 0):
      bola_deslocamento_y = 60*janela.delta_time()

    #limites y
if(bola.x > 400):
      bola_deslocamento_x = -60*janela.delta_time()

if(bola.x < 0):
      bola_deslocamento_x = 60*janela.delta_time()
```

Feito isso, ao executarmos o código veremos que a bola está ricocheteando e não mais ultrapassando as barreiras da tela.

Passo 07 - Detectando a colisão entre a bola e a barra

No nosso jogo, caso a barra toque na bola, esta irá modificar a sua direção, e irá apontar para o lado contrário.

A classe **Sprite** do **PPlay** possui um método que se chama `collided()`. Ele irá detectar a colisão entre dois objetos, e irá retornar **True**, caso um objeto colidiu com outro e **False** caso contrário.

Por exemplo, se nós tivermos:

```
barra_direita.collided(bola)
```

Teremos uma saída **True** (verdadeiro), caso o objeto `barra_direita` esteja colidindo (tocando) no objeto bola.

Vamos pensar um pouco:

- Caso a bola colida com a barra direita, isso significa que ela está se movendo para a direita (movimentação x). Sendo assim, precisamos fazer com que a bola aponte para a esquerda.
- Caso a bola colida com a barra esquerda, isso significa que ela está se movendo para a esquerda (movimentação x). Sendo assim, precisamos fazer com que a bola aponte para a direita.

A partir deste pensamento, temos a seguinte implementação:

```
if(barra_direita.collided(bola)):
        bola_deslocamento_x = -60*janela.delta_time()
if(barra_esquerda.collided(bola)):
        bola_deslocamento_x = 60*janela.delta_time()
```

Logo, caso a bola colida com a barra, ela irá modificar a sua direção.

Passo 08 - Fazendo com que o jogador ganhe pontos

Sabemos que para um jogador ganhar um ponto, a bola tem que tocar na borda do adversário. Para nos auxiliar nessa missão, vamos contar com a ajuda da classe **GameImage** do **PPlay**. O **GameImage** se parece com um **Sprite**, só que nós não podemos ter um controle a respeito da movimentação de um objeto do tipo **GameImage**. É necessário saber que a imagem que está associada a este objeto, é estática (não podemos realizar uma movimentação constante nela).

Vimos, anteriormente, uma função chamada `collided()`, ela detecta se um objeto está tocando outro. Ele foi bastante útil para ver se a bola estava tocando na barra. Esse método também está presente com objetos do tipo **GameImage**, então vai ser bastante útil para detectar se a bola está tocando na borda.

Para isso, vamos ter dois **objetos** do tipo **GameImage**, e colocaremos cada um nos cantos da direita e da esquerda da tela. Caso a bola toque no objeto que está na esquerda, o jogador da direita vai ganhar um ponto. Caso a bola toque no objeto que está na direita, o jogador da esquerda vai ganhar um ponto.

Vamos, primeiramente, realizar a importação:

```
from PPlay.gameimage import *
```

A partir da implementação, vamos criar o objeto que vai representar a borda esquerda:

```
borda_esquerda = GameImage("./assets/borda.png")
borda_esquerda.set_position(0,0)
```

Para criarmos um **objeto** do tipo **GameImage**, assim como um objeto do tipo **Sprite**, precisamos passar a localização da imagem que irá aparecer na tela.
Em seguida, estamos modificando a sua posição. O objeto que vai simbolizar a borda esquerda, irá ficar localizado no canto mais esquerdo da tela onde x vale 0 e y também.
Vamos criar agora um objeto que irá representar a borda direita, ela irá ficar no canto mais esquerdo da tela, porém como ela possui 10px de largura, estaremos descontando esses 10px da largura total da tela. Logo teremos:

```
borda_direita = GameImage("./assets/borda.png")
borda_direita.set_position(390,0)
```

Feito isso, vamos colocar o seguinte comando dentro do **game loop**:

```
borda_direita.draw()
borda_esquerda.draw()
```

Ao rodar o jogo você provavelmente não vai ver nada, pois a imagem que selecionamos para representar as bordas são da mesma cor do fundo da tela, logo ela está praticamente invisível para nós.

Vamos agora criar duas variáveis que irão guardar os pontos dos jogadores:

```
pontos_player_1 = 0
pontos_player_2 = 0
```

Dentro do **game loop**, iremos colocar algumas condições que irão verificar caso a bola esteja tocando em uma das barras:

```
if(bola.collided(borda_direita)):
        pontos_player_1+=1
        bola.set_position(200,150)

if(bola.collided(borda_esquerda)):
        pontos_player_2+=1
        bola.set_position(200,150)
```

O que os códigos estão fazendo?

- ➔ Caso a bola colida com a barra direita, o jogador 01 (da esquerda) vai ganhar 1 ponto e a bola irá reaparecer no centro.
- ➔ Caso a bola colida com a barra esquerda, o jogador 02 (da direita) vai ganhar 1 ponto e a bola irá reaparecer no centro.

Passo 09 - Mostrando os pontos na tela

Anteriormente criamos as variáveis que irão armazenar as pontuações dos jogadores. Porém, é necessário que seja realizada a sua exibição na tela. Para isso, o objeto **janela** que criamos anteriormente possui um método que irá nos ajudar. Esse método é o `draw_text()`, como a sua tradução apresenta, ele irá desenhar um texto na tela.

Esse método possui a seguinte forma:

```
janela.draw_text(texto,posicao_x,posicao_y,color=(R,G,B),font_name="nome_da_fonte",bol
d=True, italic=False)
```

Como foi apresentado acima, precisamos informar o texto que vai ser mostrado, a posição x e y em que esse texto será apresentado na tela, a sua cor no formato RGB, o nome da fonte e se ela vai se apresentar em **negrito (bold)** ou **itálico (italic)**. Uma observação a respeito do código acima: ele é escrito em somente uma linha.

Sabendo como apresentar um texto na tela, podemos fazer o seguinte, dentro do game loop, abaixo de `janela.set_background_color((255,255,255))`, iremos colocar o seguinte:

```
janela.draw_text("Pontuação",185,10, color=(0,0,0), font_name="Arial",bold=True,
italic=False)
```

Nessa linha estamos dizendo para o objeto janela para desenhar um texto que irá conter o seguinte conteúdo: “Pontuação”. Ele estará localizado nas coordenadas 185x10. Terá a cor preta. Fonte Arial. Estará em Negrito e não em Itálico.

E faremos o seguinte também:

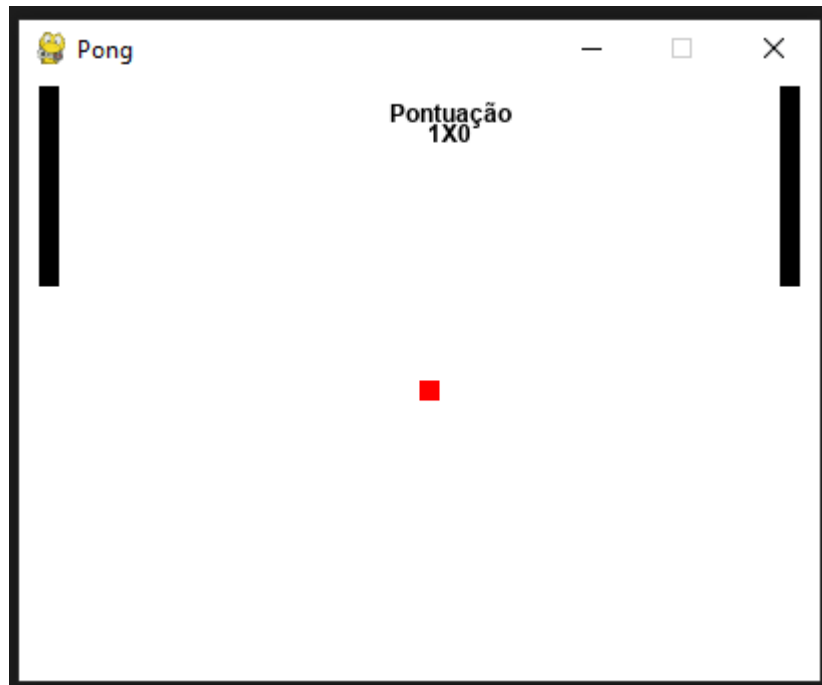
```
janela.draw_text(str(pontos_player_1)+"X"+str(pontos_player_2),204,20, color=(0,0,0),
font_name="Arial",bold=True, italic=False)
```

Neste caso, estaremos disponibilizando de fato os pontos dos jogadores na seguinte forma:

Pontos_Jogador_01 X Pontos_Jogador_02

Estamos pegando o valor que está armazenado nas variáveis que criamos anteriormente e disponibilizando na tela. Você deve ter percebido que entre as variáveis, nós temos str(). Fazemos isso pois as variáveis estão armazenando valores inteiros, porém essa função

`draw_text()` só aceita textos (Strings). Sendo assim, estamos fazendo uma conversão do valor numérico para o texto.

Ao fazer isso, teremos o seguinte:

Passo 10 - Finalizando o jogo

O nosso jogo está quase completo. Para finalizar, estaremos verificando qual jogador fez 3 pontos primeiro.
Vamos pensar como isso poderia ser feito:

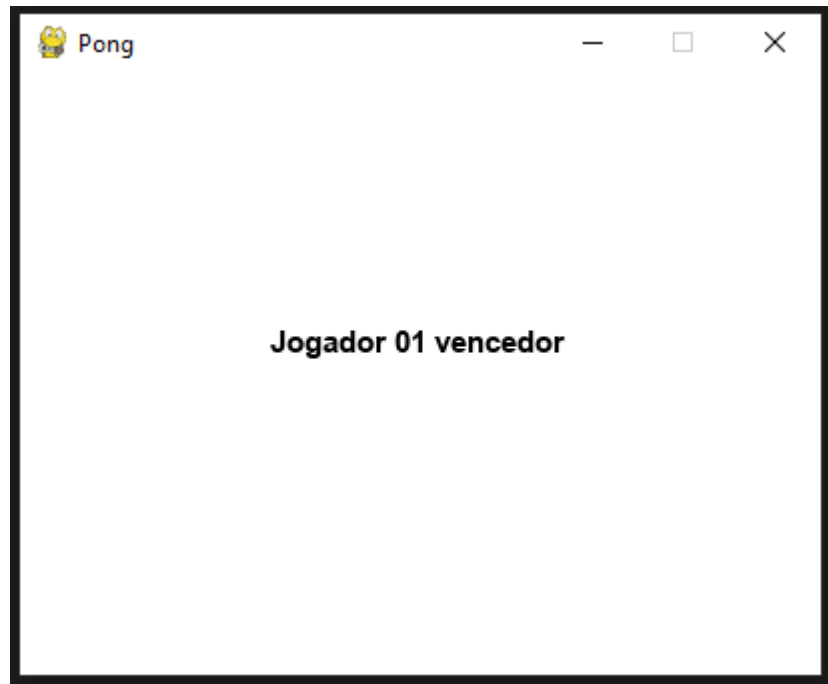
- Caso o jogador 01 (da esquerda) faça três pontos primeiro. Vamos limpar a tela (retirar todos os objetos da tela). Apresentar "Jogador 01 vencedor" e finalizar o jogo
- Caso o jogador 02 (da direita) faça três pontos primeiro. Vamos limpar a tela (retirar todos os objetos da tela). Apresentar "Jogador 02 vencedor" e finalizar o jogo.

Sendo assim, podemos colocar como a primeira coisa a ser executada no nosso **game loop** a verificação de qual jogador já completou os três pontos primeiro.
Faremos o seguinte:

```
if pontos_player_1 ==3:
        janela.clear()
        janela.draw_text("Jogador 01 vencedor",125,125, size=15, color=(0,0,0),
font_name="Arial",bold=True, italic=False)
        janela.update()
        janela.delay(5000)
        janela.close()
```

Basicamente estamos verificando se o jogador 01 fez três pontos. Caso isso seja verdadeiro, limpamos a janela, escrevemos o texto "Jogador 01 vencedor", atualizamos a tela **(isso é um detalhe importante!**). Esperamos 5000 milissegundos (5 segundos) a partir da função delay() e após isso fechamos a janela.

Podemos repetir a mesma coisa, só que com o jogador 2:

```
elif pontos_player_2 ==3:
        janela.clear()
        janela.draw_text("Jogador 02 vencedor",125,125, size=15, color=(0,0,0),
font_name="Arial",bold=True, italic=False)
        janela.update()
        janela.delay(5000)
        janela.close()
```

E assim temos o nosso primeiro jogo criado!

AULA 3 - Criando o jogo Snake

1. Sumário

Nesta aula você irá continuar na jornada de aprendizagem de criação de jogos digitais. Nas aulas passadas, começamos com o jogo **Pong**. Nesta aula, iremos começar a criar o jogo Snake.

TÓPICOS RELEVANTES

Estruturas de dados - Listas e Tuplas

Você já ouviu falar sobre estruturas de dados e porque nós o utilizamos?
Como o seu nome sugere, teremos uma estrutura para armazenar dados. Até agora, estamos armazenando os nossos dados em variáveis simples. Vejamos o seguinte problema:

"Suponha que seja necessário criar um programa que armazene os nomes de todos os estudantes cadastrados em uma escola".

Naturalmente você iria criar uma variável para armazenar esse dado:

```
nome_estudante_01 = "João"
nome_estudante_02 = "Maria"
nome_estudante_03 = "Pedro"
```

Agora imagine que você possui 100 estudantes, você iria criar uma variável para cada estudante? Seria bastante trabalhoso, correto?

Para facilitar o nosso trabalho, existe uma estrutura de dados que se chama **lista**.
Esta estrutura permite que nós possamos armazenar vários dados sob o nome de uma única variável. No nosso exemplo, com a lista, poderíamos adicionar os nomes de 100 alunos e somente ter criado uma única variável.

Como funciona uma lista?

Em python, para criar uma lista podemos fazer o seguinte:

```
lista_estudantes = ["Francisco", "Adriel","Diego","Natália","Camille","Estéfane"]
```

Podemos perceber que temos somente uma variável que é `lista_estudantes`, depois temos um operador de atribuição **(=)** e em seguida, entre **colchetes** [ ] e separados por vírgula, temos cada ítem que irá compor a nossa lista. Percebe-se que estamos armazenando strings, logo cada palavra é constituída pelas aspas.

Ao fazer o que foi mostrado acima, teremos armazenado o nome de 6 estudantes dentro de somente uma variável. Elas estão armazenadas como mostra a seguir:

posição 0	posição 1	posição 2	posição 3	posição 4	posição 5
"Francisco"	"Adriel"	"Diego"	"Natália"	"Camille"	"Estéfane"

Temos 6 itens que estão armazenados em nossa lista. Para acessar esses ítens, precisamos saber qual é o seu índice na lista. Um **índice** é basicamente a posição em que um elemento se encontra na estrutura. Na maioria das linguagens de programação, este índice começará pelo número 0 e irá incrementando o seu valor por 1 a medida em que novos itens serão adicionados na lista.

Por exemplo, se quiséssemos acessar o elemento que está na posição 4 da nossa lista, teríamos como resultado o nome "Camille".
Se quiséssemos acessar o elemento que está na posição 10 da nossa lista, teríamos um erro, pois esta lista não possui um índice com este número (o índice vai até o número 5).

Também temos a opção de criar uma lista vazia, a qual podemos inserir novos dados posteriormente:

```
lista_estudantes = []
```

Como você pode perceber, temos a variável `lista_estudantes` e após o operador de atribuição, colocamos dois colchetes que simbolizam uma lista vazia.

Sendo assim, temos os seguintes modelos gerais para a criação de uma lista em python

```
nome_variavel = []  #Cria uma lista vazia
nome_variavel = [1,2,3]  #Cria uma lista com itens
```

Como que acessamos um ítem dentro da nossa lista?

Em python precisamos fazer o seguinte:

```
print(lista_estudantes[0])
```

É necessário informar o nome da variável que está armazenando a lista de conteúdos, e sem seguida colocar entre colchetes o índice do item cujos dados se deseja obter. No caso da nossa lista de estudantes que já possuem valores inseridos, temos como saída: "Francisco".

Como adicionamos novos elementos na lista?

Devemos ter em mente que quando criamos uma lista, estamos na verdade criando um objeto do tipo **lista**. Sendo assim, podemos ter acesso a alguns métodos previamente criados (neste caso, estes métodos foram desenvolvidos pelos criadores do python).
Existe um método que se chama append. Se formos no tradutor, append significa acrescentar. Que é justamente o que precisamos neste momento.

Sendo assim:

```
lista_estudantes = ["Francisco","Adriel","Diego","Natalia","Camille","Estefane"]
lista_estudantes.append("Nadine")
print(lista_estudantes[6]) #Irá aparecer na tela "Nadine"
```

Como podemos ver, para utilizar o método append, utilizaremos a variável `lista_estudantes` que está armazenando um objeto do tipo **lista** e chamamos o método a partir do ponto (.) e em seguida o nome do método: `append`.
Para utilizarmos o método `append`, precisamos informar o que iremos inserir na nossa lista. Podemos inserir em uma lista elementos de vários tipos, números, outros objetos, palavras... Porém como estamos armazenando somente uma lista que contém somente os nomes dos estudantes, iremos continuar mantendo a padronização e iremos acrescentar o nome "Nadine".

Em seguida, iremos utilizar a função `print`, que irá mostrar na tela o novo nome inserido, que agora está na posição 6.

Como remover um item da lista?

Para remover um ítem na lista é bastante simples. Iremos utilizar o método **remove**:

```
lista_estudantes = ["Francisco","Adriel","Diego","Natalia","Camille","Estefane"]
lista_estudantes.remove("Francisco")
print(lista_estudantes[0])
```

Colocamos dentro do método remove, o que queremos remover. Neste caso, removeremos da lista o nome "Francisco". Ele irá procurar a primeira aparição deste nome e o remover (ou seja, se existissem dois "Francisco" na lista, somente um iria ser removido).

Em seguida, ao exibir na tela o ítem que está na posição 0, teremos uma reorganização da lista e "Adriel" ficará na posição 0:

posição 0	posição 1	posição 2	posição 3	posição 4	posição 5
"Adriel"	"Diego"	"Natália"	"Camille"	"Estéfane"	"Nadine"

Como percorrer todos os elementos da lista?

Já parou para pensar como podemos percorrer todos os elementos da lista? Se fosse necessário pedir para imprimir todos os nomes dos estudantes cadastrados em uma determinada turma, você iria fazer o seguinte?

```
print(lista_estudantes[0])
print(lista_estudantes[1])
print(lista_estudantes[2])
print(lista_estudantes[3])
```

O que você faria se tivéssemos 1000 estudantes cadastrados? Pense um pouco em como podemos fazer isso...

Se você pensou em utilizar uma estrutura de repetição, você pensou certo!
As estruturas de repetição serão a nossa aliada nessa jornada de percorrer os itens presentes na nossa lista. Utilizaremos a estrutura de repetição **for**:

```
for estudante in lista_estudantes:
  print("estudante")
```

O que este código está fazendo?

Estamos percorrendo a nossa lista que se chama `lista_estudantes`, e em cada vez que essa lista é percorrida, o valor do próximo elemento vai ser armazenado dentro da variável **`estudante`**. Ou seja, se temos a nossa lista com a seguinte configuração:

posição 0	posição 1	posição 2	posição 3	posição 4	posição 5
"Adriel"	"Diego"	"Natália"	"Camille"	"Estéfane"	"Nadine"

Uma vez a variável `estudante` vai ter o valor "Adriel", depois "Diego", "Natália", "Camille", "Estéfane" e por fim "Nadine". Após o último valor da lista ser pego, o **for** irá ser finalizado e a lista não será mais percorrida.
No nosso caso, estamos dando um **print** na variável `estudante`, que irá assumir os valores previamente informados, os quais estão presentes na nossa lista.

Uma estrutura de dados imutável: a tupla.

Vamos apresentar agora um outro tipo de estrutura de dados: **a tupla**.
Ela é bem parecida com uma lista, porém possui uma grande diferença: uma vez que os seus valores foram atribuídos, eles não podem ser alterados. Ou seja, após a sua atribuição, não é possível adicionar, remover ou alterar os valores dentro dessa estrutura.

Vamos ver como podemos criar uma tupla:

```
coordenadas = (-12.2733, -38.9556)
```

Acabamos de criar uma estrutura de dados que está armazenando uma posição de um determinado lugar. Na primeira posição desta lista (no índice 0) temos a sua coordenada x, e na segunda posição desta lista (no índice 1) temos a sua coordenada y.
Temos que ter em mente que essas posições **não podem ser alteradas!**

Percebemos que, diferentemente de uma lista, ao criar uma tupla, nós utilizamos os parênteses em vez dos colchetes.

Logo, assim como a lista, a tupla irá ajudar a ter menos variáveis no nosso programa:

```
longitude = -12.2733
latitude = -38.9556

coordenadas = (-12.2733, -38.9556)
```

Em vez de termos duas variáveis, uma que irá armazenar a longitude e outra a latitude, temos somente uma que irá armazenar uma tupla das coordenadas de um ponto no globo.

Embora os elementos de uma tupla não possam ser alterados, uma variável que guarda uma tupla pode ser alterada, passando a guardar outra tupla em lugar da anterior.

```
coordenadas = (-12.2733, -38.9556)
coordenadas = (-13.003587, -38.5121857).
```

Acessando os elementos de uma tupla:

Assim como a lista, iremos acessar os seus elementos através do seu índice:

```
coordenadas = (-12.2733, -38.9556)
print(coordenadas[0]) # -12.2733
print(coordenadas[1]) # -38.9556
```

Percorrendo os elementos de uma tupla:

Assim como a lista, também podemos percorrer todos os elementos de uma tupla, utilizando a estrutura de repetição **for**:

```
coordenadas = (-12.2733, -38.9556)
for coordenada in coordenadas:
   print(coordenada)
```

Revisão de Funções

O jogo Pong, o qual criamos na última aula, era organizado na forma de um único programa principal. Caso seja necessário adicionar novas funcionalidades, podemos ter alguns problemas:

- O programa principal pode ficar muito grande
- Poderemos ter repetições de trechos de códigos

Como podemos resolver esses problemas?
Se você pensou na **modularização (criação de funções)**, você pensou corretamente!

Em Python, nós podemos criar **módulos** de código que irão desempenhar alguma tarefa específica. Estes módulos são chamados de funções.

Basicamente, estes módulos serão subprogramas que serão executados quando necessário. Para executar um subprograma, o programador precisa chamá-lo.

Vamos pensar em um exemplo:

Quais são os passos que você faz todo dia pela manhã?
A maioria das pessoas segue o seguinte fluxo:

1. Levantar da cama
2. Tomar banho
3. Tomar café
4. Escovar os dentes
5. Ir para a escola

Todos os dias você segue esse mesmo conjunto de passos a fim de realizar uma tarefa: se arrumar para a escola. Os dias vão se passando, e você continua fazendo os mesmos passos:

1. Levantar da cama
2. Tomar banho
3. Tomar café
4. Escovar os dentes
5. Ir para a escola

Vamos supor que você deseja construir um programa no qual você vai criar um calendário mensal, onde a cada dia você terá que colocar a sua rotina.
Você poderia ter algo desse tipo:

```
print("DIA 01"
print("06:00 - Acordar")
print("06:10 - Tomar banho")
print("06:30 - Tomar café")
print("06:50 - Escovar os dentes")
print("07:00 - Ir para a escola")
# Aqui vai ter os outros passos que você faz no dia
print("22:00 - Dormir")

print("DIA 02"
print("06:00 - Acordar")
print("06:10 - Tomar banho")
print("06:30 - Tomar café")
print("06:50 - Escovar os dentes")
print("07:00 - Ir para a escola")
# Mais códigos
print("22:00 - Dormir")

print("DIA 03"
print("06:00 - Acordar")
print("06:10 - Tomar banho")
print("06:30 - Tomar café")
print("06:50 - Escovar os dentes")
print("07:00 - Ir para a escola")
```

Você percebe que temos uma repetição de código? Imagina quantas linhas iriam ter se você tivesse mais passos antes de ir para a escola. E imagine quantas linhas o seu código total terá contando com os outros dias.

Em python, podemos fazer o seguinte:

```
def rotina_manha():
    print("DIA 01")
    print("06:00 - Acordar")
    print("06:10 - Tomar banho")
    print("06:30 - Tomar café")
    print("06:50 - Escovar os dentes")
    print("07:00 - Ir para a escola")

print("DIA 01")
rotina_manha()
# Aqui vai ter os outros passos que você faz no dia

print("DIA 02")
rotina_manha()

print("DIA 03")
rotina_manha()
```

Perceba como o código diminuiu bastante. Antes, tínhamos uma repetição de um conjunto de ações (os prints que mostram a rotina da manhã) para acabar com a repetição, realizamos um agrupamento destes comandos em um único lugar (a nossa função)

Em seguida, no nosso programa principal, chamamos esta função o qual irá ocupar somente 1 linha. Posteriormente, você pode perceber, que podemos realizar os mesmos passos ao chamar o nome da função.

Vamos supor que no café da manhã você possui uma grande variedade de comidas. A cada dia você irá comer algo diferente. Sendo assim, ao chamar a função `rotina_manha` temos que indicar de alguma forma que iremos comer uma comida específica. Veja o seguinte:

```
def rotina_manha(cafe_da_manha):
    print("DIA 01")
    print("06:00 - Acordar")
    print("06:10 - Tomar banho")
    print("06:30 - Tomar café "+cafe_da_manha)
    print("06:50 - Escovar os dentes")
    print("07:00 - Ir para a escola")

print("DIA 01")
rotina_manha("Café com biscoito") # Comer café com biscoito

print("DIA 02")
rotina_manha("Cereal") #Comer cereal

print("DIA 03")
rotina_manha("Bolo de milho") #Comer bolo de milho
```

Perceba que a função mudou um pouco. Entre parênteses, temos uma variável chamada `cafe_da_manha`, que, a depender da chamada da função, irá receber um determinado valor. Como podemos ver nas chamadas da função, primeiramente colocamos “Café com biscoito” na outra chamada colocamos “Cereal” e por fim temos “Bolo de milho”.

O que foi mostrado acima, simboliza a passagem de parâmetros de uma função. Na passagem de parâmetros, estamos passando algum conteúdo do código principal para dentro da função. Podemos passar qualquer tipo de conteúdo para a função, uma string, uma lista, a possibilidade é bastante ampla.

Vamos supor que agora você quer verificar se a partir do dia, você terá aula à tarde. Por exemplo, se o dia da semana for quarta-feira, o seu programa vai ter que retornar que você possui aula à tarde.

```
def rotina_manha(cafe_da_manha):
    print("DIA 01")
    print("06:00 - Acordar")
    print("06:10 - Tomar banho")
    print("06:30 - Tomar café "+cafe_da_manha)
    print("06:50 - Escovar os dentes")
    print("07:00 - Ir para a escola")


def aula_a_tarde(dia_da_semana):
    if(dia_da_semana=="Quarta"):
        return "Possuo aula pela tarde"
    else:
        return "Não possuo aula pela tarde"

print("DIA 01")
rotina_manha("Café com biscoito") # Comer café com biscoito
retorno_funcao = aula_a_tarde("Quarta")
print(retorno_funcao) #Possuo aula pela tarde

print("DIA 02")
rotina_manha("Cereal") #Comer cereal

retorno_funcao = aula_a_tarde("Quinta")
print(retorno_funcao) #Não possuo aula pela tarde
```

Criamos uma função chamada `aula_a_tarde`, ela irá verificar a variável `dia_da_semana` que recebe por parâmetro. Se o valor desta variável for "Quarta", ele terá que retornar um dado para o programa principal. Neste caso, o dado a ser retornado é "Possuo aula pela tarde". Caso contrário, ele irá retornar: "Não possuo aula pela tarde". Este retorno do dado só é possível graças à palavra especial chamada return.

No programa principal, quando chamamos a função, percebemos que temos uma atribuição de um valor a variável `retorno_funcao`. Estamos fazendo isso, pois a função irá retornar um dado, e uma das melhores formas de armazenar este dado é com uma variável. Sendo assim, em seguida apresentamos o valor desta variável na tela. Na primeira chamada a mensagem "Possuo aula pela tarde" será exibida, e nas seguintes "Não possuo aula pela tarde".
Lembre-se. É bastante importante chamar a função dentro do programa principal, pois caso você não a chame, ela **nunca** será executada.

Vamos ver um exemplo simples:

```
def exibe_mensagem():
    print("Hello world")

exibe_mensagem()
print("Olá mundo!")
```

Qual mensagem você acredita que será exibida na tela?
Neste caso teremos “Hello World” e em seguida “Olá mundo!”.

```
def exibe_mensagem():
    print("Hello world")

print("Olá mundo!")
```

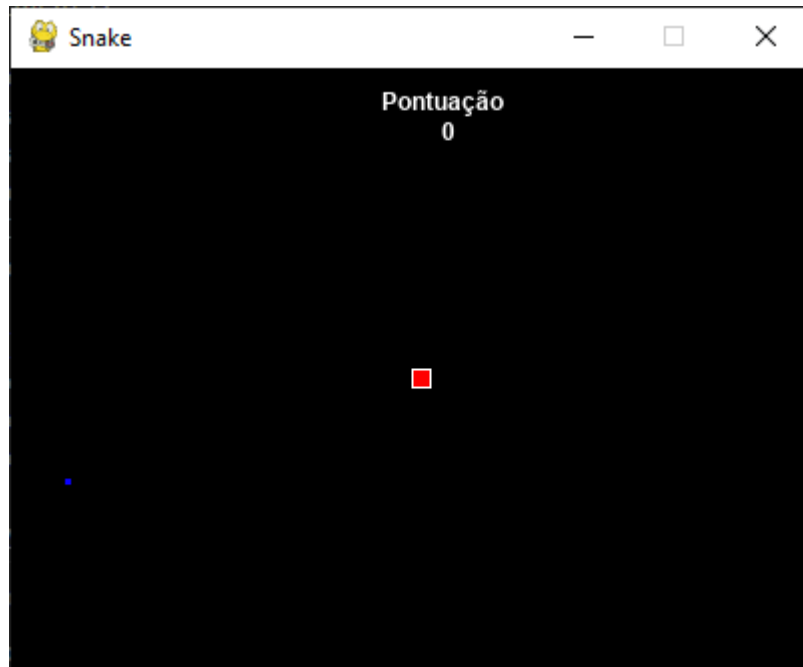
Já neste caso, somente a mensagem “Olá mundo!” será exibida na tela, pois não estamos chamando a função `exibe_mensagem`!

Vimos como podemos utilizar uma função. Vamos ver como é a sua forma geral:

```
def nome_da_funcao(parametros):
    #Conteudo

#No programa principal, fora do escopo da função, você tem que a
chamar:

nome_da_funcao(parametros)
```

Temos a palavra reservada **`def`** que define uma função. Em seguida, apresentamos o seu nome. Depois, entre parênteses, iremos apresentar os parâmetros (variáveis e valores) que essa função irá receber. E por fim, no programa principal, não podemos deixar de chamar a função para ela ser executada.

Você sabia que o `print()` é uma função? Ela recebe como parâmetro a string que você deseja que seja exibida na tela.
Além disso, temos outra função que deve ser conhecida por você, a função `input()` recebe valores do usuário.

FOLHA DE ATIVIDADES - Criando o jogo Snake

Este é outro jogo clássico. Também conhecido com o jogo da cobrinha, o snake pode ter feito parte da infância de muitas pessoas, pois o jogo esteve presente na maioria dos celulares de antigamente. Ele foi criado inicialmente em 1976 e popularizado nos anos 90.
Basicamente temos uma cobra, que a partir da comida que irá coletar ela irá crescendo. O jogo termina quando a cabeça da cobra bate na borda do jogo ou no seu corpo..

Inicialmente, será necessário baixar os arquivos importantes para a criação do jogo. Acesse o seguinte link: http://bit.ly/ArquivoSnake

Extraia-os em uma pasta de sua escolha.

Em seguida, nesta mesma pasta, no diretório principal, crie um arquivo chamado **snake.py**. Nele iremos implementar o nosso código.

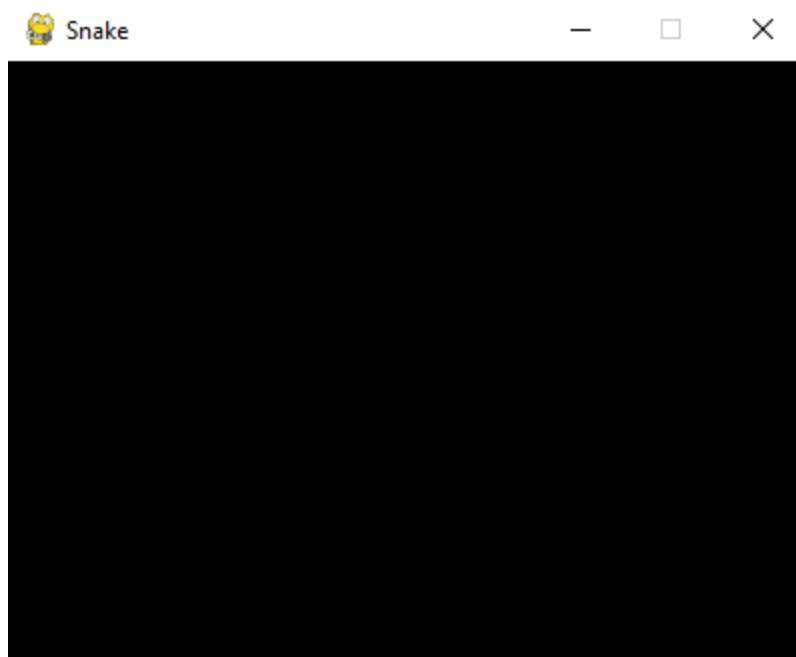
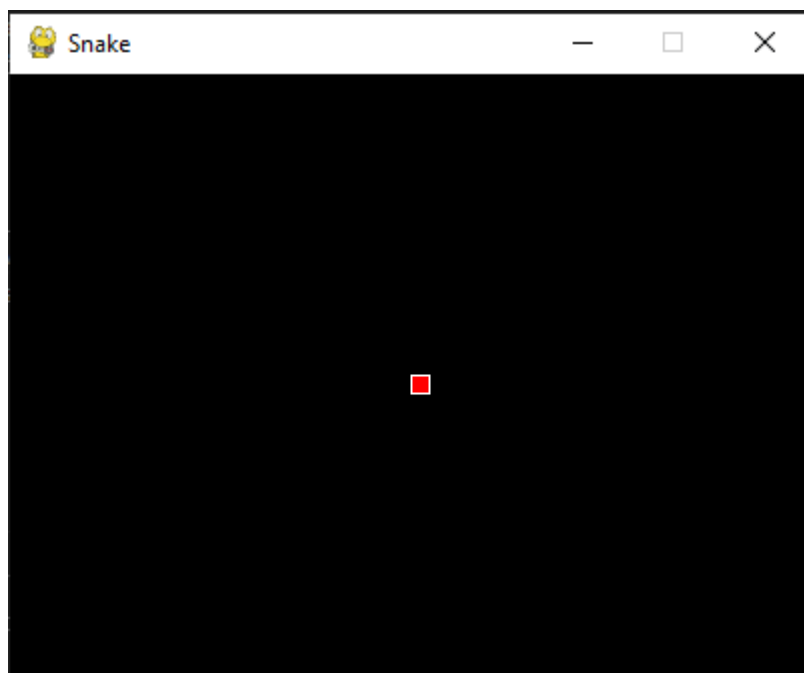
Passo 01 - Criando a tela do jogo

Assim como foi feito no jogo anterior, será necessário criar a tela por onde o nosso jogo vai acontecer, vamos ter o seguinte:

```
from PPlay.window import *
janela = Window(400,300)
janela.set_title("Snake")

while True:
    janela.set_background_color((0,0,0))
    janela.update()
```

Relembrando o que já foi feito na aula anterior, temos a importação necessária da classe Window, que vem com o **PPlay**. Em seguida, criamos o objeto do tipo Window, e informamos que essa janela terá 400 pixels de largura e 300 pixels de altura.
Modificamos, também, o título da janela que irá ser “Snake” (mas se você quiser também pode chamar de jogo da cobrinha).

A partir da linha 6 teremos o nosso **game loop**, no qual o nosso jogo irá rodar. Dentro dele informamos que a cor de fundo da nossa tela será preto (utilizando o sistema de cores **RGB**). E por fim, na linha 9 chamamos o método que atualiza as informações da janela.

Com esse código, temos o seguinte:

Passo 02 - Adicionando a cobra no jogo

Isso é o que queremos!

Como vocês imaginam que podemos fazer isso? Lembre-se do que fizemos no jogo anterior.

Assim como no jogo anterior, utilizaremos a classe Sprite do **PPlay**:

```
[...]

from PPlay.sprite import *
cabeca = Sprite('./assets/snake.png')
cabeca.set_position(200,150)

while True:
    janela.set_background_color((0,0,0))
    cabeca.draw()
    janela.update()
```

No nosso jogo Snake, temos que a movimentação central do jogo será a partir da cabeça da cobra, sendo assim, temos a variável cabeça que irá armazenar um objeto do tipo Sprite, a partir dela, iremos realizar a sua movimentação.

Passo 03 - Movimentando a cobra.

Para movimentar a cobra, utilizaremos os comandos do teclado, assim como fizemos no jogo anterior. Desta vez, utilizaremos as teclas de cima, baixo, esquerda e direita do teclado.

Vamos realizar as importações necessárias:

```
from PPlay.keyboard import *
[...]
teclado = Keyboard()
```

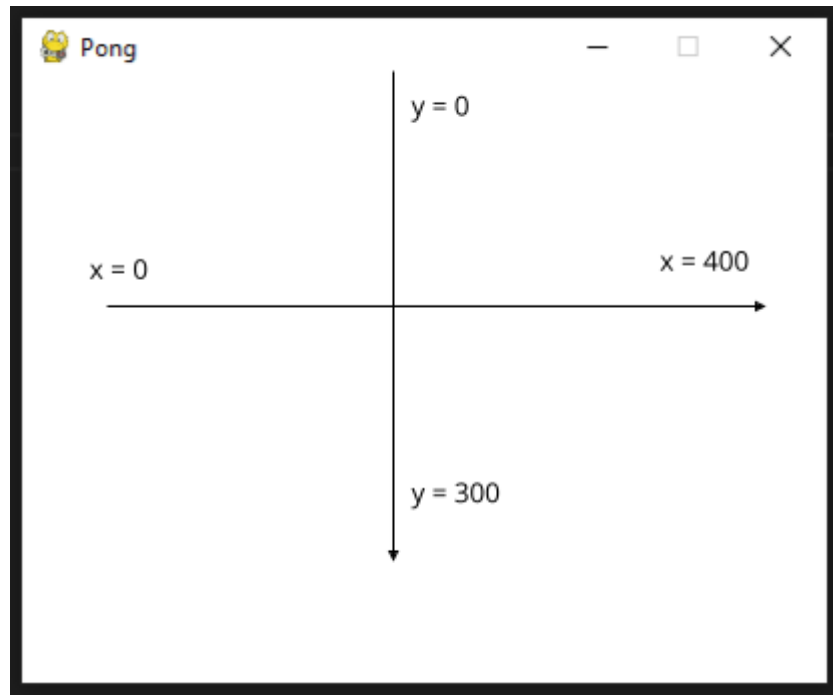
Desta vez, criaremos uma tupla que será responsável por armazenar as coordenadas x e y da cabeça da cobra. Ela será bastante importante para a movimentação do jogo:

```
direcaoCobra = (0,0)
```

Não se preocupe com a inicialização com (0,0). Como vocês devem saber, não podemos inicializar uma tupla com valores vazios. A cada momento em que a cobra se movimenta, iremos criar uma nova tupla e atribuir essa tupla a variável `direcaoCobra`.

Em seguida, dentro do **game loop,** podemos colocar o seguinte:

```
if(teclado.key_pressed("UP")):
   direcaoCobra = (0,-100)
if(teclado.key_pressed("LEFT")):
   direcaoCobra = (-100,0)
if(teclado.key_pressed("DOWN")):
   direcaoCobra = (0,100)
if(teclado.key_pressed("RIGHT")):
   direcaoCobra = (100,0)
```

Você se lembra de como funciona a tela no **PPlay**? Caso não se lembre, veja a imagem a seguir:

Até agora você deve ter percebido que a cobra ainda não está se movimentando de fato. Nós só estamos modificando o valor da variável `direcaoCobra`. Para fazermos com que a cora se movimente de fato, iremos criar uma função chamada `movimentacao_cobra`. Ela irá ser responsável pela movimentação da cobra em geral:

```
def movimentacao_cobra():
        cabeca.move_x(direcaoCobra[0]*janela.delta_time())
        cabeca.move_y(direcaoCobra[1]*janela.delta_time())
```

Você deve estar se preocupando com o porquê de criar uma função que irá ter somente duas linhas simples. Não se preocupe, logo logo ela irá aumentar
Em seguida, dentro do game loop, chamaremos a função de movimentar a cobra e de desenhar a cabeça:

```
movimentacao_cobra()
cabeca.draw()
```

Ao fazer isso, a sua cobra irá se movimentar!

Passo 04 - Deixando o movimento da cobra menos suave

Se tudo estiver bem, a cobra está se movimentando de uma forma bem suave. Vamos deixar esse movimento um pouco menos suave, para lembrar mais da forma clássica do jogo.
Você se lembra da explicação sobre FPS? Atualmente o nosso jogo está rodando com uma taxa de 60 frames por segundo. Vamos diminuir essa taxa e fazer com que o jogo rode com apenas 10 frames por segundo.

Para isso, vamos realizar a seguinte importação:

```
from pygame.time import *
[...]
clock = Clock()
```

Na primeira linha temos a importação do módulo que estaremos utilizando. Este módulo vem direto do **pygame** que é por onde o **PPlay** funciona. Após as importações que você já tem no seu código, vamos criar um novo objeto do tipo Clock. Ele irá nos ajudar com a mudança da taxa de quadros por segundo.

Feito isso, dentro do seu **game loop**, na primeira linha logo após o **while**, coloque o seguinte:

```
clock.tick(10)
```

Com isso, você irá perceber que a cobra está se movimentando de uma forma menos suave, bem parecida com o jogo clássico.

Passo 05 - Inserindo a comida da cobra no jogo

Um item necessário no nosso jogo é a comida a qual a cobra irá se alimentar para posteriormente crescer. Ela irá aparecer em um lugar aleatório dentro da área visível do jogo. Para isso, vamos fazer a importação do pacote que nos ajuda a gerar números aleatórios:

```
import random
```

Como anteriormente já importamos a classe Sprite do **PPlay**, iremos agora só criar o objeto do tipo Sprite que armazenará a imagem da comida:

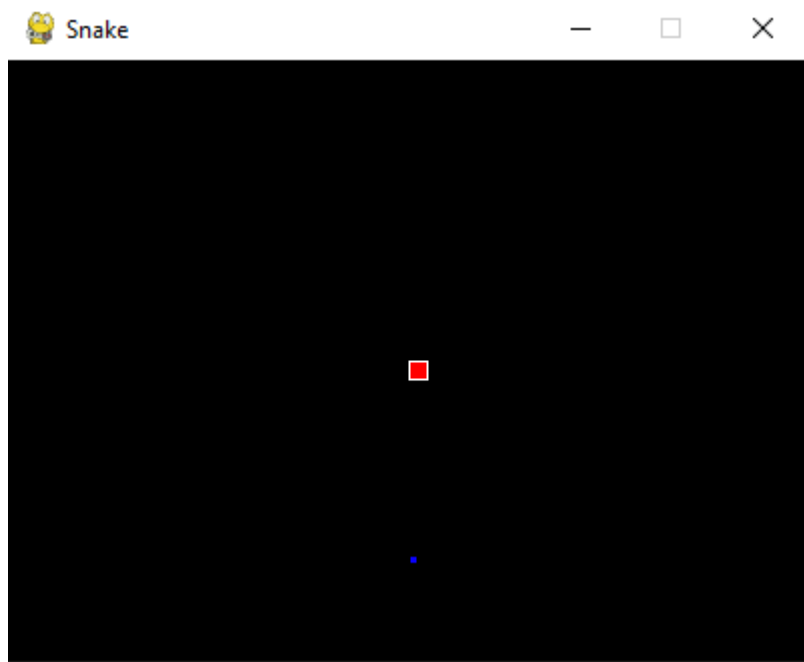
```
comida = Sprite('./assets/food.png')
comida.set_position(random.randint(0,390),random.randint(0,290))
```

Estamos utilizando o método `set_position` para alterar as variáveis x e y de uma só vez.
A imagem da comida possui 10px de largura e 10px de altura. Já o tamanho total da nossa tela é de 400 px de largura e 300 px de altura. Será necessário descontar 10 pixels de cada coordenada (x,y) para que a imagem apareça dentro da área visível da nossa tela. Sendo assim, para a posição x, estamos gerando um número aleatório entre 0 e 390. E para posição y, geramos um número aleatório entre 0 e 290.

Por fim, dentro do nosso **game loop**, podemos colocar o seguinte:

```
comida.draw()
```

Feito isso, a comida irá aparecer em um local aleatório na tela:

Snake

Passo 06 - Fazendo a cobra comer a comida

Por agora, iremos fazer com que quando a cobra toque na comida, ela desapareça e em seguida apareça em um outro lugar aleatório.

Isso vai ser bem simples, dentro do game loop teremos o seguinte:

```
if(cabeca.collided(comida)):
    comida.set_position(random.randint(0,390),random.randint(0,290))
```

Por enquanto iremos fazer isso. Quando a cobra tocar na comida, ela irá aparecer em outro lugar.

AULA 4 - Concluindo o jogo Snake

1. Sumário

Nesta aula você irá continuar a construção do jogo Snake que foi iniciado na aula passada.

FOLHA DE ATIVIDADES - Continuando o jogo Snake

Passo 01 - Crescendo a cobra

Este é o passo mais complexo do nosso jogo, o qual envolve o crescimento da cobra e como a cobra irá se comportar após o seu crescimento.

Para isso, vamos dividir em algumas subseções:

- ➔ Aumentando de fato o tamanho da cobra
- ➔ Movimentação da cobra com o novo tamanho

Aumentando de fato o tamanho da cobra:

Vamos pensar o seguinte: quando a cobra aumenta de tamanho, temos que deixar isso visível na tela para que o jogador possa saber que ela está maior. Atualmente podemos disponibilizar imagens na tela a partir de Sprites. Não podemos criar variáveis simples para armazenar objetos de Sprites pois não sabemos qual vai ser o maior tamanho que a cobra vai ter. Sendo assim, utilizaremos uma estrutura de dados já conhecida: a **lista**.

```
corpo_sprites = []
```

Esta lista irá se iniciar vazia pois quando o jogo começa, a cobra ainda está pequena.
Criaremos, agora, uma outra lista. Esta nova lista irá armazenar as posições da cabeça e do corpo da cobra. Essa lista será bastante importante para fazermos a movimentação do corpo da cobra:

Vamos agora implementar o método que irá fazer com que essa lista comece a aumentar de tamanho:

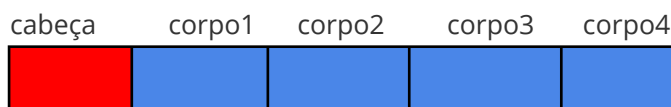
```
def aumentar(): #adiciona um novo corpo na lista de corpos
    corpo_sprites.append(Sprite('./assets/snake.png'))
```

Feito isso, precisamos chamar este método que acabamos de criar. Ele será chamado toda vez que a cobra comer a comida. Vamos alterar o código que já fizemos anteriormente, e ele vai ficar assim:

```
if(cabeca.collided(food)):
    food.set_position(random.randint(0,400),random.randint(0,300))
    aumentar()
```

Se você executar o seu código, vai notar que a cobra não está crescendo. De fato, a gente não está fazendo com que a cobra cresça visualmente. E é o que iremos fazer agora a partir da movimentação da cobra

Movimentando a cobra

Como funciona a movimentação da cobra?
Podemos pensar em um sistema de troca de posições, vamos pensar o seguinte:
Atualmente nós temos o controle da cabeça da cobra, nós podemos realizar os movimentos de acordo com a entrada do usuário a partir do teclado. Vejamos o seguinte esquema:

A imagem acima representa a nossa cobra, vamos supor que as coordenadas x e y da cabeça da cobra são: 100px e 100px.

Em seguida, nós temos uma cobra um pouco grande. Na nossa lista `corpo_sprites` temos 4 Sprites que armazenam a imagem do corpo da cobra, simbolizando o seu crescimento.

As coordenadas x e y do objeto `corpo1` são, respectivamente: 110px e 100px.
As coordenadas x e y do objeto `corpo2` são, respecitvamente: 120px e 100px.
As coordenadas x e y do objeto `corpo3` são, respectivamente: 130px e 100px.
As coordenadas x e y do objeto `corpo4` são, respecitvamente: 140px e 100px.

Até aqui tudo bem, temos que a cobra está em uma determinada posição e o seu corpo está a sua direita.

O que acontece quando a cabeça da cobra se movimenta? Você concorda que o corpo tem que seguir a cabeça?

Quando a cabeça da cobra se movimentar, esta cabeça irá assumir um novo valor. Porém para simular que o corpo está se movimentando, iremos pegar a posição anterior que a cabeça da cobra estava e passar para o `corpo1`. Assim como vamos passar a posição anterior que o objeto corpo1 tinha e passar para o `corpo2`, e assim vai:

Vejamos a tabela:

Coordenadas durante o tempo (x,y)	Cabeça da cobra	corpo1	corpo2	corpo3	corpo4
1	100px, 100px	110px, 100px	120px,100px	130px,100px	140px, 100px
2	90px,100px	100px,100px	110px,100px	120px,100px	130px,100px
3	80px,100px	90px,100px	100px,100px	110px,100px	120px,100px
4	70px,100px	80px,100px	90px,100px	100px,100px	110px,100px
5	60px,100px	70px,100px	80px,100px	90px,100px	100px,100px

Você percebe que somente a cabeça da cobra está obtendo novos valores de coordenada? E as suas coordenadas atuais são passadas para os objetos de que armazenam o Sprite de corpo.

Como você imagina que podemos implementar essa funcionalidade?

Se você pensou em algo envolvendo listas e a estrutura de repetição for, você pensou corretamente!

Vamos primeiro criar uma lista que irá armazenar todas as posições da cobra (incluindo a cabeça). A primeira posição da lista irá conter uma tupla que armazena as coordenadas da cabeça, e as próximas posições irão conter tuplas contendo as coordenadas do corpo da cobra (caso ela cresça):

```
cobra_posicoes = [(200,150)]
```

Temos que a primeira posição é a tupla contendo as coordenadas iniciais da cabeça da cobra: 200px e 150px.

Vamos agora modificar uma função que criamos anteriormente, a `movimentacao_cobra()`. Vamos seguir os seguintes requisitos:

- ➔ Vamos fazer as permutações dos valores das coordenadas entre os corpos dentro da lista que armazena as posições da cobra.
- ➔ Movimentar o corpo da cobra com as novas coordenadas
- ➔ Modificar o valor das coordenadas da cabeça da cobra (simulando a movimentação da cobra). E salvar esse novo valor dentro da lista de coordenadas

Atualmente, temos o seguinte:

```
def movimentacao_cobra():
        cabeca.move_x(direcaoCobra[0]*janela.delta_time())
        cabeca.move_y(direcaoCobra[1]*janela.delta_time())
```

Inicialmente vamos desconsiderar o que fizemos por enquanto, logo em seguida este código voltará para a função.

Vocês lembram da tupla que criamos anteriormente que armazena a posição atual da cobra? Se você não se lembra, ela se chama: `direcaoCobra`. Inicialmente ela começa com o seguinte valor:

```
direcaoCobra = (0,0)
```

Você concorda que a cobra só irá se movimentar, caso o valor dessa tupla seja diferente de (0,0)? Sendo assim, a primeira coisa que iremos fazer no nosso procedimento é colocar o seguinte:

```
def movimentacao_cobra():
    if(direcaoCobra!=(0,0)):
        #iremos completar posteriormente
```

Então a cobra irá se movimentar caso a sua posição seja diferente de (0,0). Ou seja, caso ela não esteja parada.

Para realizarmos a permutação de vetores, iremos utilizar a estrutura de repetição for. Como você acredita que pode ser realizada essa permutação dos valores?

Uma das possíveis soluções que podemos fazer é a seguinte:

```
def movimentacao_cobra():
    if(direcaoCobra!=(0,0)):
        for pos in range(len(cobra_posicoes),-1,0,-1):
            cobra_posicoes[pos] = cobra_posicoes[pos-1]
```

Neste trecho, estamos percorrendo a lista de posições de trás para frente. E para cada item percorrido, fazemos as trocas das posições.

Feito isso, as posições do corpo da cobra estão atualizadas e podemos passar essas posições da lista para o objeto do corpo. Lembrando, temos que as coordenadas estão presentes somente na lista de coordenadas, precisamos passar essas coordenadas para o corpo da cobra para posteriormente chamarmos o método draw().

Vamos criar uma função para realizar essa ação:

```
def movimentar_corpo():
    for i in range(1,len(cobra_posicoes)):
        corpo_sprites[i-1].x = cobra_posicoes[i][0]
        corpo_sprites[i-1].y = cobra_posicoes[i][1]
```

Estamos utilizando o laço de repetição for para percorrer a lista que armazena as posições da cobra. Estamos começando o laço a partir do índice 1, pois o índice 0 armazena as coordenadas cabeça, e precisamos das coordenadas do corpo.

Iremos chamar esse procedimento que acabamos de criar, dentro da função que estávamos criando (movimentacao_cobra):

```
def movimentacao_cobra():
    if(direcaoCobra != (0,0)):
        for pos in range(len(cobra_posicoes)-1,0,-1):
            cobra_posicoes[pos] = cobra_posicoes[pos-1]
        movimentar_corpo() #Movimenta o corpo
```

Para finalizar temos que movimentar o corpo da cobra com as novas coordenadas e modificar o valor das coordenadas da cabeça da cobra (simulando a movimentação da cobra). E salvar esse novo valor dentro da lista de coordenadas:

Faremos o seguinte:

```
def movimentacao_cobra():
    if(direcaoCobra != (0,0)):
        for pos in range(len(cobra_posicoes)-1,0,-1):
            cobra_posicoes[pos] = cobra_posicoes[pos-1]
        movimentar_corpo()
        cabeca.move_x(direcaoCobra[0]*janela.delta_time())
        cabeca.move_y(direcaoCobra[1]*janela.delta_time())
        cobra_posicoes[0] = (cabeca.x, cabeca.y)
```

Feito isso, temos o nosso método de movimentação da cobra feito com sucesso!

Vamos agora chamar esse procedimento dentro do nosso **game_loop.**

Vamos rodar o nosso jogo. Deu tudo certo?

Se ainda não estiver mostrando o corpo da cobra, fique tranquilo que isso é para acontecer mesmo!
Algumas coisas estão faltando:

Adicionar uma lista vazia dentro das listas de posições quando a cobra cresce:

```
def aumentar():
    corpo_sprites.append(Sprite('./assets/snake.png'))
    cobra_posicoes.append([])
```

Ao fazer isso, garantimos que a permutação das coordenadas também se aplique a esse novo objeto que foi adicionado.

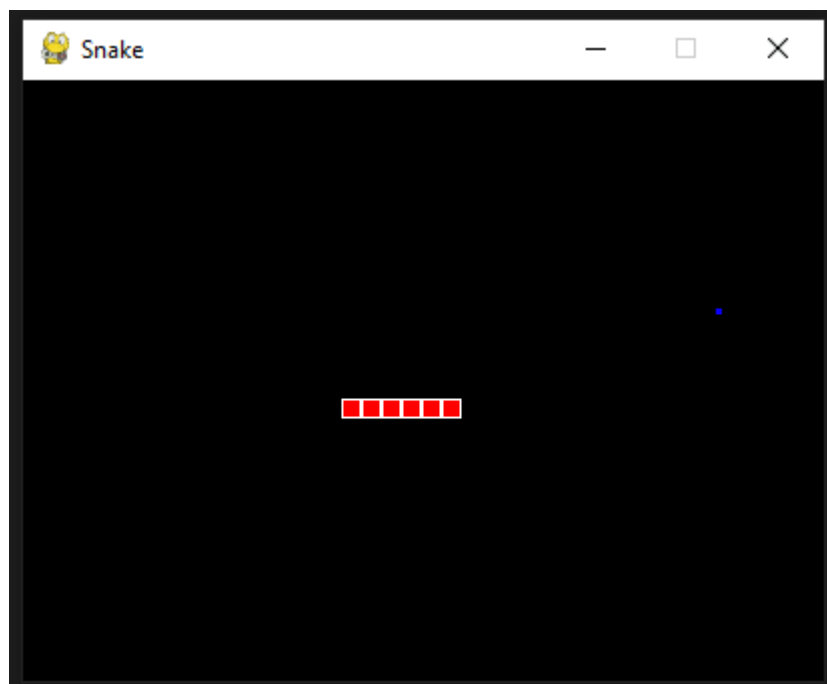
Chamar o método draw para cada corpo existente na lista de corpos:

Sabemos que para desenharmos um objeto do tipo Sprite na tela do jogo, precisamos chamar o método `draw()`. Temos uma lista que armazena os objetos do tipo Sprite, a `corpo_sprites`. Para cada objeto presente nesta lista, temos que chamar o método draw, para ele aparecer na tela.

Sendo assim, vamos criar uma função para realizar isto:

```
def desenhar_corpo():
    for corpo in corpo_sprites:
        corpo.draw()
```

E por fim, chamar a função no **game_loop**

Ao fazer isso, vamos ter o seguinte:

Passo 02 - Disponibilizando o total de pontos

Estamos quase chegando no final do nosso jogo. Precisamos agora disponibilizar na tela, o total de pontos que o jogador possui. Calcularemos esse total de pontos a partir do tamanho da lista que armazena os sprites do corpo da cobra:

Dentro do nosso **game loop**, teremos isso:

```
    janela.draw_text("Pontuação",185,10, color=(255,255,255), font_name="Arial",bold=True, italic=False)
    janela.draw_text(str(len(cobra_posicoes)-1),215,25, color=(255,255,255), font_name="Arial",bold=True, italic=False)
```

Feito isso, teremos:

Passo 03 - Finalizando o jogo

Bom, o nosso jogo está quase finalizado.
O jogo só termina quando a cobra bate nas bordas do jogo, sendo assim, teremos que fazer um código que verifique se a cabeça da cobra está tocando nas bordas da tela e caso isso aconteça, a tela de fim de jogo (assim como o do jogo anterior) irá aparecer:

Vamos fazer isso em uma função:

```
def verifica_fim_jogo():
    if( (cabeca.y >300 or cabeca.y < 0) or (cabeca.x > 400 or cabeca.x < 0)):
        janela.clear()
        janela.draw_text("Você        perdeu!",150,150,        size=15,        color=(0,0,0),
font_name="Arial",bold=True, italic=False)
        janela.draw_text("Pontuação    Total:    "+str(len(corpo_sprites)),140,170,    size=15,
color=(0,0,0), font_name="Arial",bold=True, italic=False)
         janela.update()
         janela.delay(5000)
         janela.close()
```

Feito isso, não esqueça de chamar esta função dentro do **game loop** e o jogo estará finalizado!

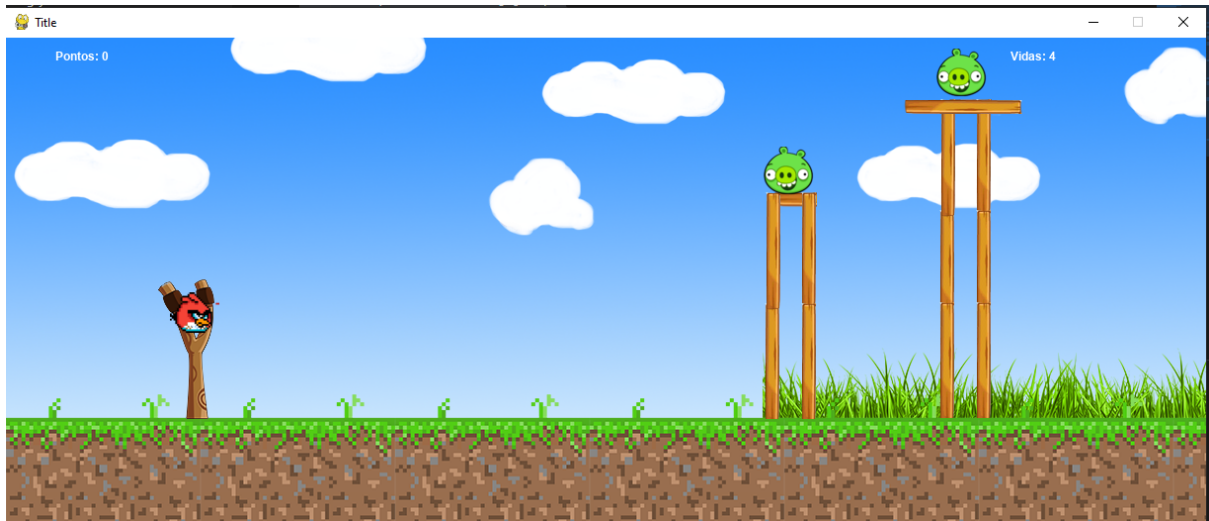
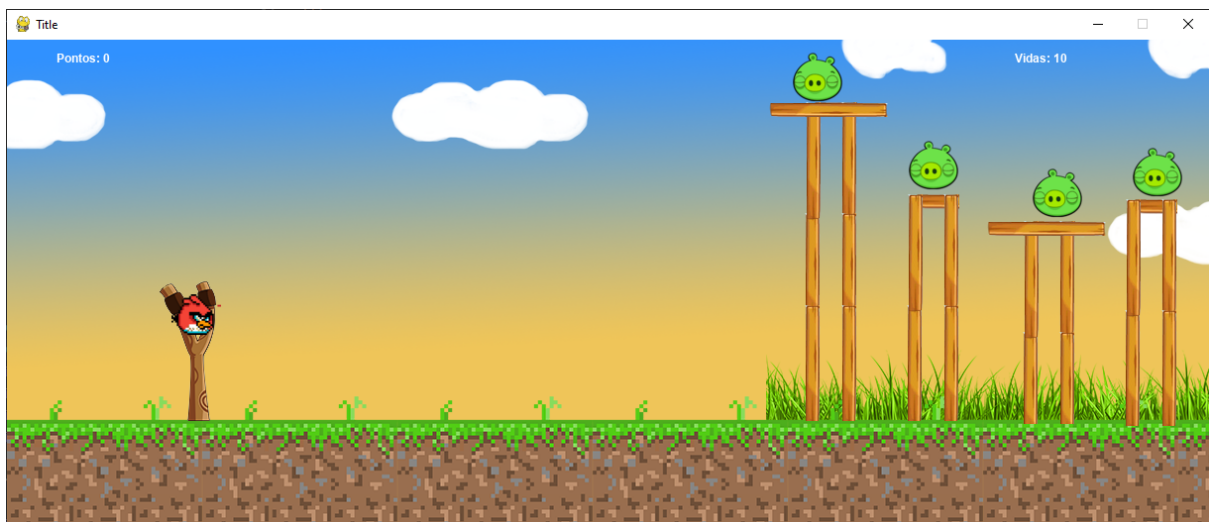
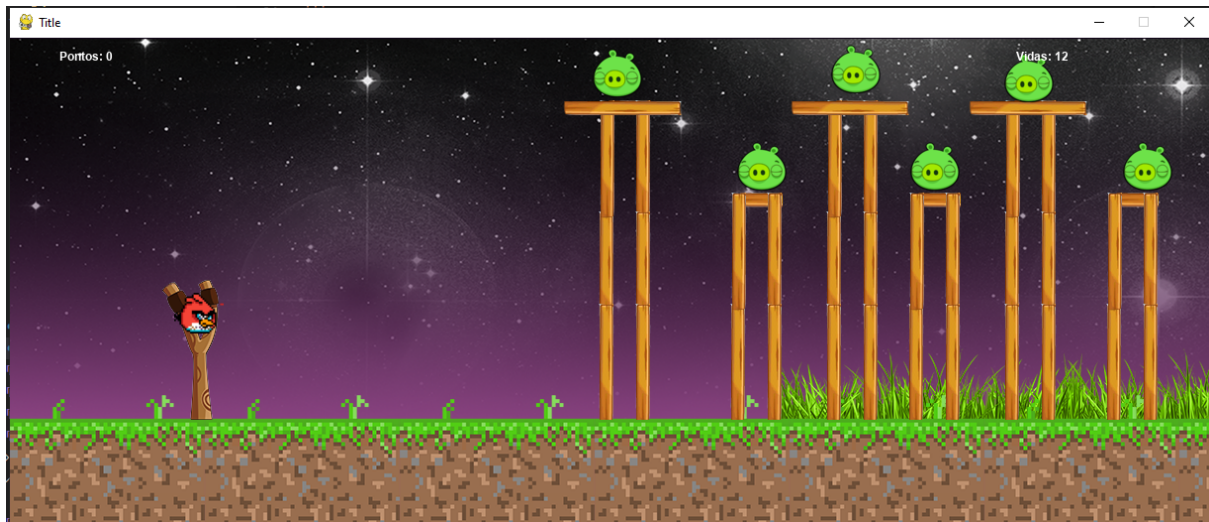
Uma sugestão para implementação: pense em como o jogo pode estar finalizando também quando a cabeça da cobra bate no seu corpo!

AULA 5 - Construindo o jogo Angry Birds.

1. Sumário

Nesta aula iremos continuar nas construções dos jogos digitais utilizando **Python** e **PPlay**.

FOLHA DE ATIVIDADES - Criando o jogo Angry Birds

Hoje iremos começar a construir o nosso novo jogo: o famoso Angry Birds:
O nosso jogo vai ter três fases, como veremos a seguir:

Fase 01 do jogo

Fase 02 do jogo

Fase 03 do jogo

Este jogo será um pouco mais complexo que os outros. Ele terá três fases e o nível de dificuldade entre estas fases irá aumentar.

Vamos utilizar uma abordagem diferente para esta aula. Inicialmente iremos colocar os Sprites na tela do jogo e em seguida iremos adicionar as suas funcionalidades.

Neste momento você já deve saber muito sobre a instanciação de objetos e como criar uma tela. Sendo assim, vamos deixar você tentar deixar o seu código como a imagem da primeira fase.

Inicialmente, baixe os arquivos necessários para o projeto: http://bit.ly/ArquivoAngryBirds

Passo 01 - Conhecendo as imagens do jogo

As imagens necessárias vão estar dentro da pasta */assets*. A tela de jogo (objeto `janela` terá as seguintes dimensões: (1215px,494px)

Para o primeiro nível, vamos ter as seguintes imagens:

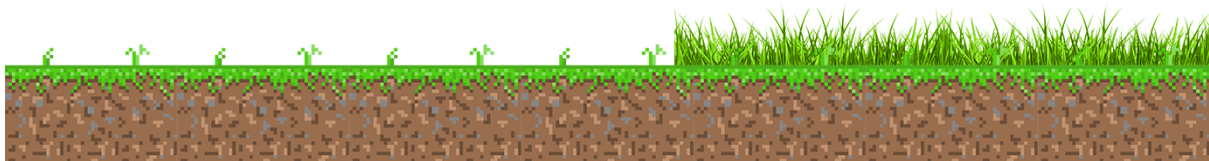
A Imagem de fundo que irá ficar na tela (dica: use o GameImage)

O estilingue em que o pássaro vai ser armazenado.

Imagem do porco

Imagem do pássaro

Plataforma número 01 onde o porco irá ficar

Plataforma número 02 onde o porco irá ficar.

Você deve ter reparado que o porco possui 3 imagens, estas estão simulando um movimento. Podemos animar um sprite a partir da imagem acima. Temos que em cada pedaço de imagem o porco realiza um movimento. Primeiro ele está com os olhos abertos, em seguida ele fecha e por fim, ele abre.
Vamos fazer o seguinte:

```
porco1 = Sprite('porco.png',3)
```

Antes, ao criar um objeto do tipo Sprite, a gente só informava o caminho da imagem. Agora, além de informar o caminho da imagem, informamos a quantidade de quadros que a imagem vai ter, que no caso é 3.

Passo 02 - Fazendo a sua parte!

Agora que você sabe como vai ser a estrutura base do nosso jogo. Agora é a sua vez de codificá-lo. Utilize os conhecimentos que você obteve com os jogos anteriores! Lembre-se de um fato importante: o jogo terá três fases então vamos ter um código bem modularizado, para cada fase terá um cenário diferente.

Esperamos que ao rodar o jogo, esteja tudo funcionando corretamente!

Vamos ver uma forma de construir este cenário:

```
from PPlay.window import *
from PPlay.sprite import *
from PPlay.gameimage import *
from pygame.time import *


janela = Window(1215,494)
mouse = janela.get_mouse()
passaro = Sprite('./assets/passaro.png')
passaro.set_position(166,257)

fundo = GameImage('./assets/level01bg.png')
chao = GameImage('./assets/grass.png')
chao.y= 313
estilingue = GameImage('./assets/slingshot.png')
estilingue.set_position(142,235)

plataformas = []
porcos = []
vidas = 4
pontos = 0
```

```
def construir_mundo_nivel_1():
    plataforma1 = GameImage('./assets/platform1.png')
    plataforma1.set_position(760,145)

    plataforma2 = GameImage('./assets/platform2.png')
    plataforma2.set_position(900,54)

    plataformas.append(plataforma1)
    plataformas.append(plataforma2)

    porco1 = Sprite('./assets/porco.png',3)
    porco1.set_position(765,110)
    porco1.set_total_duration(900)

    porco2 = Sprite('./assets/porco.png',3)
    porco2.set_position(940,11)
    porco2.set_total_duration(900)


    porcos.append(porco1)
    porcos.append(porco2)
```

```
def desenha_porcos():
    for porco in porcos:
        porco.draw()
        porco.update()


def desenha_plataformas():
    for plataforma in plataformas:
        plataforma.draw()

passaro.set_position(166,257)
construir_mundo_nivel_1()

while(True):
    fundo.draw()
    chao.draw()
    estilingue.draw()
    desenha_plataformas()
    passaro.draw()
    desenha_porcos()
                janela.draw_text("Vidas:     "+str(vidas),1017,12,     color=(255,255,255),
font_name="Arial",bold=True, italic=False)
                janela.draw_text("Pontos:     "+str(pontos),50,12,     color=(255,255,255),
font_name="Arial",bold=True, italic=False)
    janela.update()
```

Esta é a estrutura principal do nosso jogo. Não se preocupe se você tiver feito algo diferente do que está sendo mostrado aqui. O mais importante deste código é ter uma função que cuide somente da criação dos elementos da tela daquele nível. Esta função vai ser bastante importante para mantermos a organização do código.

Passo 03 - Utilizando o mouse

Se você já jogou *Angry Birds* anteriormente, você sabe que o lançamento do pássaro no estilingue é realizado a partir do mouse. Ao clicar e segurar com o botão esquerdo do mouse, podemos esticar o pássaro e ao soltar o botão, o pássaro é lançado.

Como estamos fazendo um jogo mais simples, não iremos simular o movimento do estilingue. Porém iremos utilizar o mouse para definir uma nova posição para onde o pássaro vai ser lançado.

Vamos agora no nosso **game loop** fazer um código que faz com que o pássaro siga o ponteiro do mouse:

Já criamos um objeto do tipo Window e o armazenamos na variável `janela`. Um objeto do tipo Window armazena todas as informações necessárias para a janela do jogo. Como já vimos, a partir deste objeto podemos verificar qual tecla o usuário pressionou.
Temos o seguinte:

```
janela = Window(1215,494)
mouse = janela.get_mouse()
```

Abaixo da criação do objeto do tipo Window que está sendo armazenada na variável `janela`, estamos utilizando o método `get_mouse()` que irá nos retornar um novo objeto que é responsável pelo mouse. (Isso mesmo, a partir de um objeto, podemos ter acesso a outro objeto - isso só foi possível graças ao criador do **PPlay** que fez essa implementação).

Um método bastante importante que está armazenado na variável mouse é o `get_position()`. Quando chamado, ele irá retornar uma tupla contendo as posições x e y do mouse. Vamos fazer o seguinte. Dentro do nosso **game loop** escreva o seguinte:

```
print(mouse.get_position())
```

Rode o seu jogo e passe o mouse na tela. Você irá ver no terminal que terá as coordenadas atuais do mouse. A primeira posição é a coordenada x e a segunda, a coordenada y. Algo desse tipo:

```
(986, 463)
(992, 454)
(999, 440)
(1004, 426)
(1005, 416)
(1005, 404)
(983, 378)
(945, 351)
(888, 326)
(829, 302)
(773, 287)
(713, 279)
```

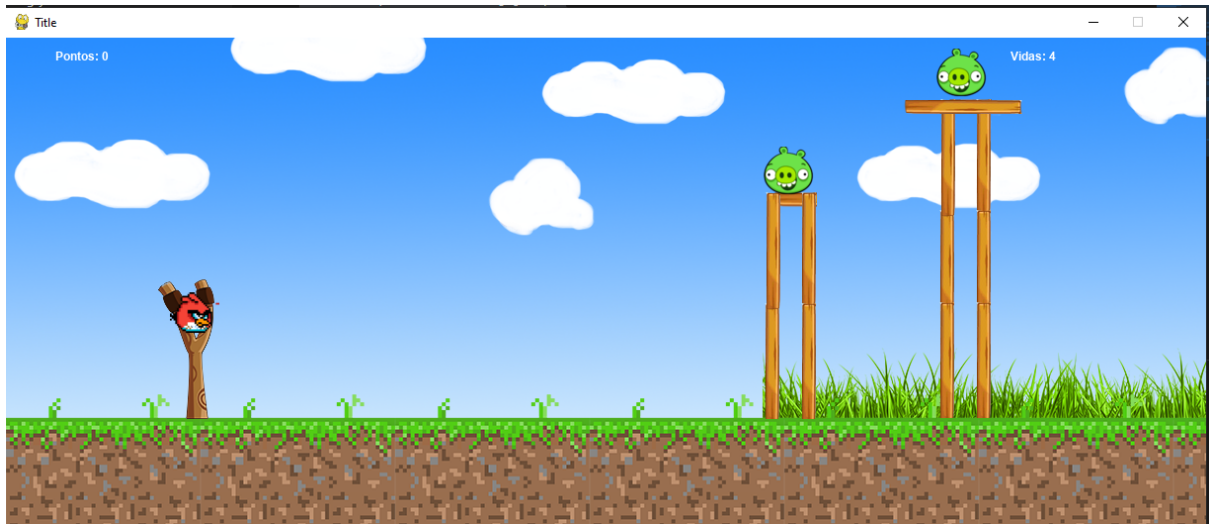
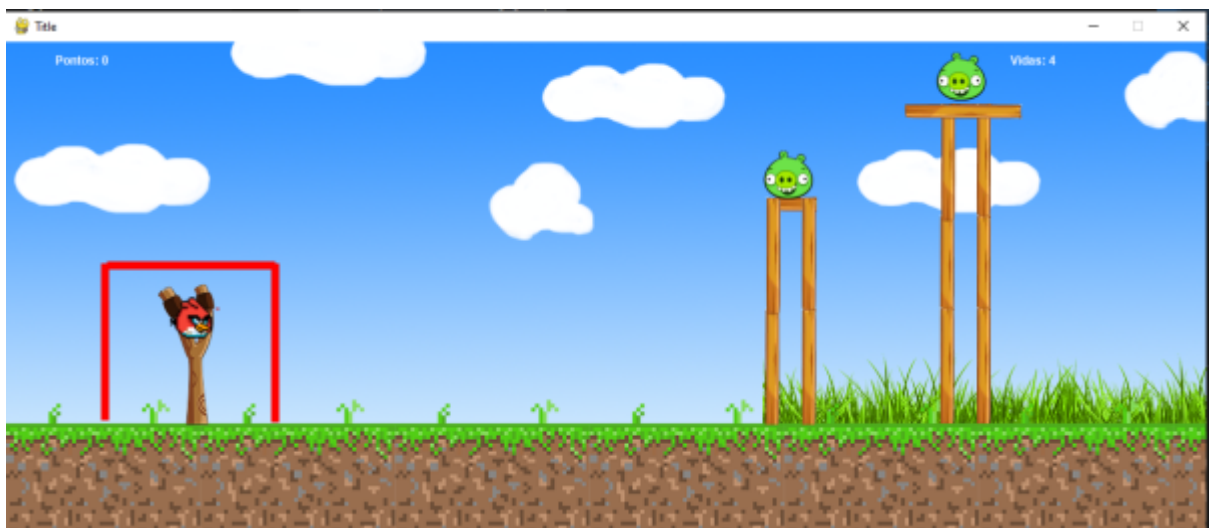
Após fazer esse teste, você pode apagar esse print.

Vamos fazer agora com que enquanto você move o mouse, a imagem do pássaro irá se movimentar seguindo o ponteiro. Você imagina como podemos fazer isso?

Se você pensou em colocar o seguinte dentro do **game loop**:

```
coordenada_x_mouse = mouse.get_position()[0]
coordenada_y_mouse = mouse.get_position()[1]

passaro.set_position(coordenada_x_mouse, coordenada_y_mouse)
```

Passo 04 - Limitando a movimentação do pássaro

Até agora o nosso pássaro está se movimentando livremente na janela. Não é uma coisa que queremos pois ele será lançado a partir do estilingue. Vamos limitar a sua movimentação. Podemos pensar o seguinte:

Esta é a nossa tela de jogo. Vamos imaginar que existe um quadrado imaginário onde o estilingue está contido. Sendo assim, o pássaro só vai poder se movimentar por dentro deste "quadrado":

Ou seja, teremos que fazer uma verificação para ver se o pássaro está dentro das coordenadas que são consideradas adequadas, ou seja, dentro do "quadrado" imaginário para ele se mover:

Vamos criar uma função que faz essa verificação:

```
def passaro_pode_mover(coordenada_x,coordenada_y):
    if (12 <= coordenada_x <=203) and (220 <= coordenada_y <= 380):
        return True
    else:
         False
```

Essa função recebe dois valores: as coordenadas x e y do pássaro. E caso as suas coordenadas x e y estejam entre um determinado valor que define se o pássaro pode se mover, a função tem um retorno verdadeiro. Caso contrário, falso.

Vamos chamar a função dentro do nosso **game loop**:

```
if(passaro_pode_mover(coordenada_x_mouse,coordenada_y_mouse)):
    passaro.set_position(coordenada_x_mouse, coordenada_y_mouse)
```

Agora, caso o pássaro esteja perto do estilingue, ele irá poder se movimentar!

AULA 6 - Continuando o jogo Angry Birds

1. Sumário

Nesta aula iremos continuar a construção do jogo Angry Birds que iniciamos na aula passada.

TÓPICOS RELEVANTES

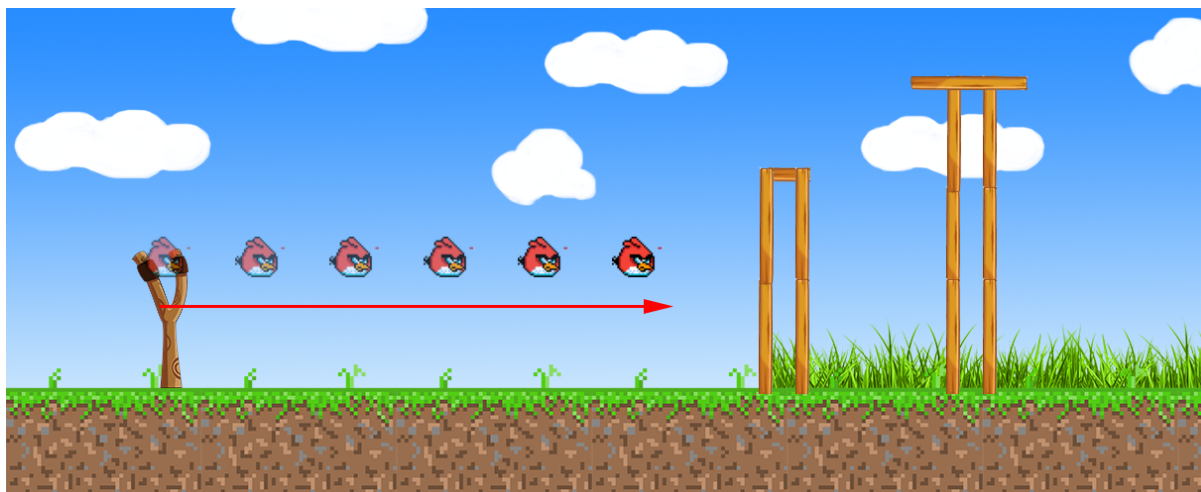
Lançamento Oblíquo

Para sabermos como funciona o lançamento do pássaro no jogo, temos que entender um pouco sobre o **Lançamento Oblíquo**. Esse assunto da física está muito presente no jogo.

Neste lançamento, temos a junção do movimento vertical e o movimento horizontal. Ou seja, o corpo ao ser lançado neste tipo de movimento, terá as coordenadas x e y alteradas ao mesmo tempo. Para entendermos melhor este lançamento, vamos conhecer melhor o **movimento horizontal** e o **lançamento vertical para cima**:

Para entendermos este lançamento oblíquo, vamos dar uma olhada em cada um dos movimentos que o compõem.

Movimento Horizontal

Este tipo de movimento, também chamado de movimento uniforme, ocorre quando um corpo é movimentado com velocidade **constante**. Como este corpo terá a sua velocidade constante, a única variável que terá mudanças, será a sua **posição**.

Como vemos na imagem acima, ocorre um deslocamento na coordenada **x** do objeto durante o **tempo**.

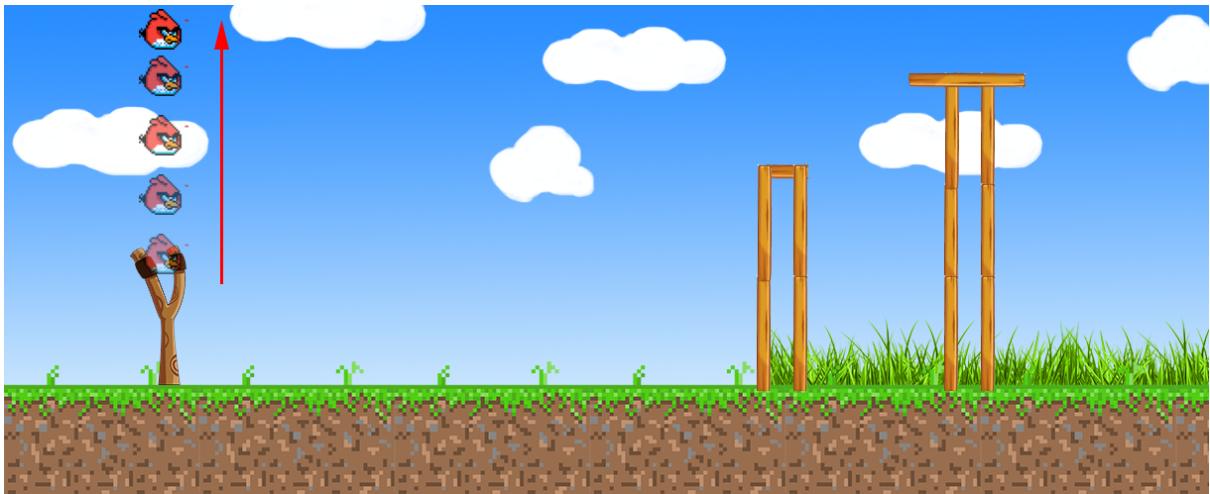
Temos como fórmula do movimento horizontal o seguinte:

$$\mathbf{x(t) = x_0 + v_o t}$$

Esta fórmula também é conhecida como **função horária do movimento uniforme**.

Ou seja, caso desejamos obter a posição (coordenada x) de um determinado objeto em um certo tempo t (maior que 0, afinal o tempo 0 é quando o movimento não ocorreu) teremos

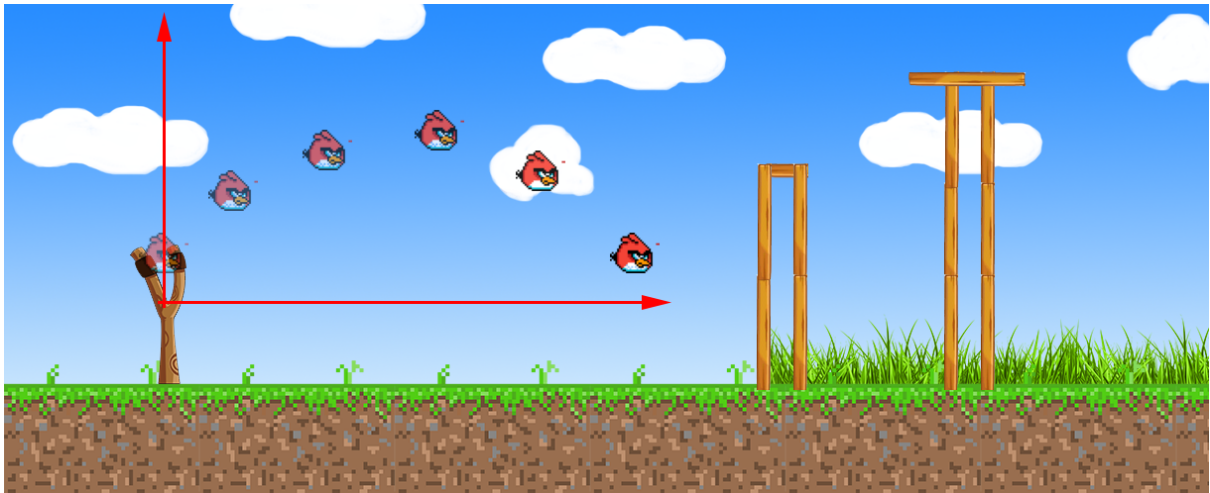
Movimento Vertical

Neste tipo de movimento, temos um parâmetro bastante importante: a gravidade. Quando lançamos um objeto para cima, percebemos que a decorrer da distância que ele percorre, ele vai perdendo a sua velocidade até que em um momento ele começa a descer. (Você já fez esse experimento? Se não, faça uma tentativa).

A partir deste experimento, podemos perceber que nesse tipo de movimento, ao contrário do anterior, **não possui uma velocidade constante**:

$$y(t) = y_o + v_o t - gt^2/2$$

Esta fórmula é a função horária da posição.Se você perceber, ela é bastante parecida com a função horária do movimento uniforme, só que temos a presença do campo: $gt^2/2$.
Esta parte indica que a gravidade vai atuar como uma força contrária ao movimento, perceba que existe o sinal de menos. Então, ao lançar um objeto para cima, ele irá ter uma velocidade inicial v_o porém, com o passar do tempo, essa velocidade vai diminuindo por causa da gravidade, até que ela se torne zero. Quando isso acontece, o corpo irá começar a cair.

Lançamento Oblíquo

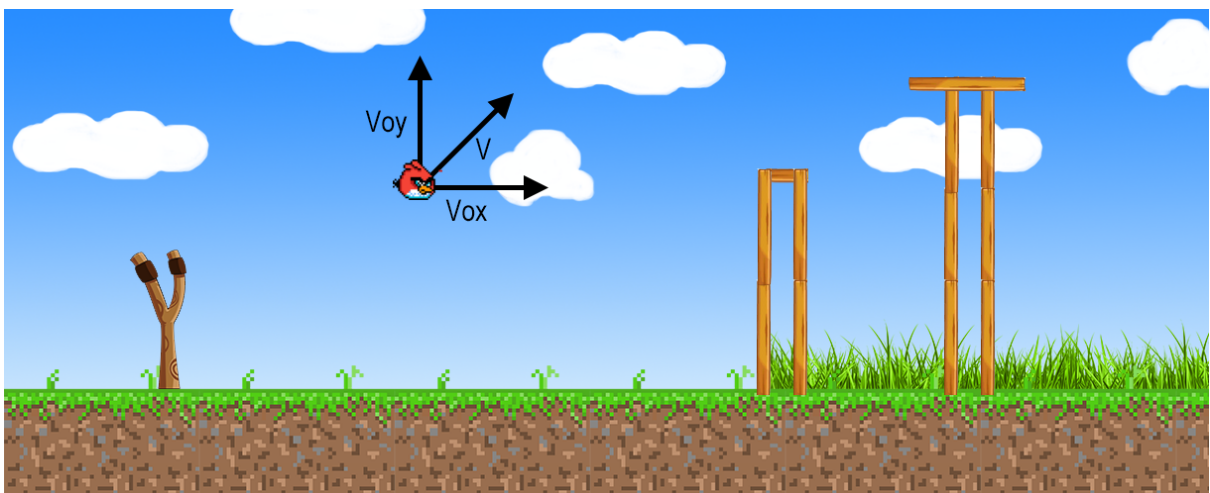
Podemos, agora, juntar os dois movimentos que acabamos de conhecer para formar o lançamento oblíquo.

Veja a imagem do canhão. Quando a bola é lançada, ela terá um deslocamento nas suas coordenadas x (caracterizando o movimento horizontal) e y (caracterizando o movimento vertical). Como teremos duas velocidades, uma constante no movimento horizontal e outra que varia no movimento vertical, podemos ter a seguinte representação:

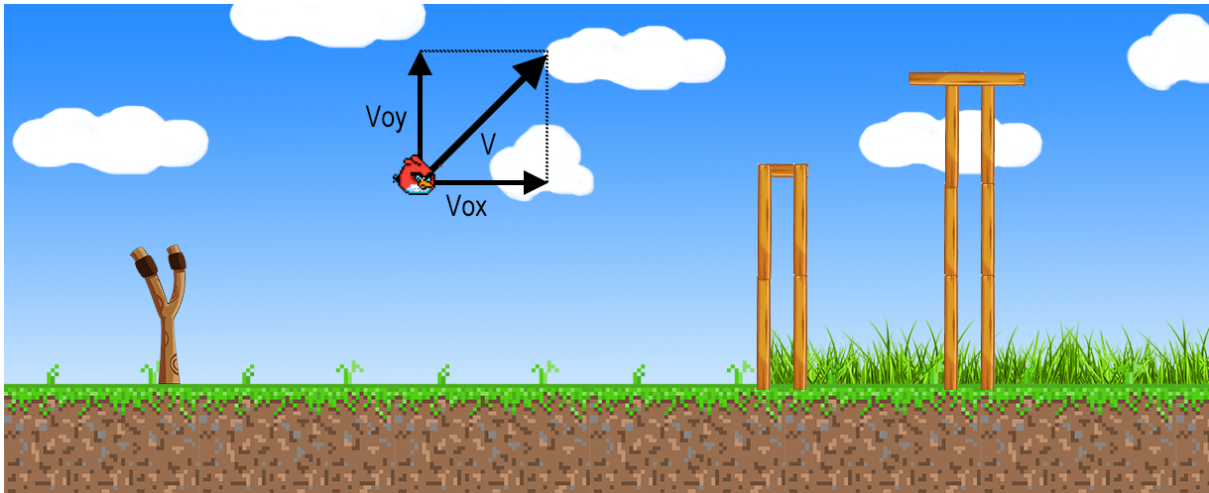
$$\mathbf{x(t) = x_o + v_o xt} \text{ ,}$$

e,

$$\mathbf{y(t) = y_o + v_o yt - gt^2/2}$$

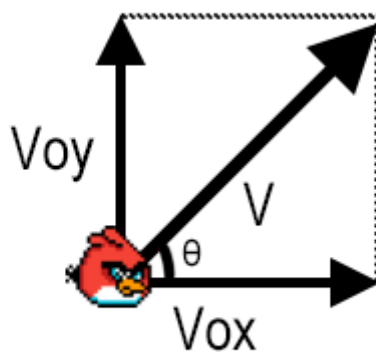
Estas são as mesmas fórmulas que você viu anteriormente, só trocamos a nomenclatura da velocidade de cada movimento para não confundirmos.

Podemos relacionar as duas velocidades utilizando a geometria analítica, veja a imagem a seguir:

Se você conhece como funcionam os vetores, já deve estar imaginando aonde iremos chegar. Caso você não conheça, não se preocupe.

Temos a presença de um triângulo retângulo:

Pitágoras (570 a.C - 495 a.C) criou um teorema no qual relaciona os lados de um triângulo retângulo. Logo, a partir da imagem acima, podemos ter o seguinte:

$$v_o^2 = v_ox^2 + v_oy^2$$

Sendo assim, se for necessário obter o módulo da velocidade inicial de um corpo podemos calcular a soma das velocidades iniciais de cada movimento (horizontal e vertical).

Caso tenhamos o módulo da velocidade inicial de um corpo (v_o), podemos também calcular a velocidade inicial em cada movimento (vertical ou horizontal).
Utilizando alguns artefatos da física, podemos realizar uma **projeção** como mostra a imagem a seguir:

E teremos o seguinte:

$$v_{ox} = v_o \cos(\theta)$$
$$v_{oy} = v_o \text{sen}(\theta)$$

FOLHA DE ATIVIDADES - Continuando o jogo Angry Birds

Passo 01 - Lançando o pássaro

Anteriormente foi dito como seria o funcionamento do lançamento do pássaro, vamos relembrar:
Ao clicar e segurar com o botão esquerdo do mouse, podemos esticar o pássaro e ao soltar o botão, o pássaro é lançado.

Ou seja, o pássaro só vai ser esticado no estilingue a partir do clique no botão esquerdo do mouse e após o usuário deixar de segurar este botão, ele é lançado.
Para nos ajudar com isso, o objeto `mouse` possui o método `mouse.is_button_pressed()` que recebe como parâmetro qual botão do mouse se deseja fazer a verificação.

- ➔ 1 - Botão esquerdo
- ➔ 2 - Botão do meio
- ➔ 3 - Botão direito

Teremos um retorno **True** caso aquele botão está sendo pressionado, e **False** caso contrário:

```
print(mouse.is_button_pressed(1)) #Retorno True

print(mouse.is_button.pressed(2)) #Retorno False
```

Caso o botão a ser pressionado no momento for o da esquerda, temos como retorno **True**.

A partir do conhecimento deste novo método, vamos agora restringir o movimento do pássaro para somente quando o botão esquerdo do mouse for pressionado.
Qual lógica você utilizaria?

Se você pensou em algo neste estilo:

```
if(mouse.is_button_pressed(1)):
        coordenada_x_mouse = mouse.get_position()[0]
        coordenada_y_mouse = mouse.get_position()[1]
        if(passaro_pode_mover(coordenada_x_mouse, coordenada_y_mouse)):
            passaro.set_position(coordenada_x_mouse, coordenada_y_mouse)
```

Está tudo certo! (Mas não se preocupe se você fez de outra forma, se está tudo funcionando, então tudo bem!)

Agora o pássaro só consegue se movimentar dentro do limite previamente estabelecido e caso o botão esquerdo esteja pressionado.

Vamos agora definir a velocidade de lançamento do pássaro. Como **vimos nos Tópicos Relevantes**, o lançamento oblíquo é a junção do movimento horizontal com o vertical.
No jogo original, a depender do quão esticado o pássaro é, mais longe ele irá. Vamos implementar essa funcionalidade. Na física, temos que a distância de um corpo é proporcional a sua velocidade. Ou seja, quanto maior a velocidade de lançamento de um corpo, maior é a distância que ele irá percorrer e vice-versa.

Respeitando a modularização do nosso código, vamos criar uma função que irá tratar da movimentação do pássaro:

```
def movimentar_passaro(posicao_x, posicao_y):
```

Perceba que, para o pássaro se movimentar, será necessário passar as suas posições iniciais x e y.
Sabemos que o pássaro irá possuir um movimento oblíquo (junção do movimento vertical e horizontal). Como vimos nos **Tópicos Relevantes**, a velocidade vertical não é constante, diferentemente da velocidade horizontal. Sendo assim, ela estará sofrendo a ação da “gravidade” constantemente: o pássaro irá subir e graças a ação da “gravidade” ela irá cair.

Vamos primeiro fazer o movimento vertical do pássaro. Iremos utilizar a função horária da posição:

$$y(t) = y_o + v_{oy}t - gt^2/2$$

Temos que:
y_o - é a posição inicial do pássaro - ela será obtida a partir da coordenada y do mouse.
Voy - é a velocidade inicial do pássaro, teremos que achar uma maneira de calcular essa velocidade.
t - representa o tempo que se passou desde o lançamento
g - é uma constante que define a gravidade.

Vamos começar com a criação de uma variável que irá armazenar o valor da gravidade:

```
GRAVIDADE = 330
```

Perceba que essa variável foi declarada com as letras maiúsculas. Essa declaração é somente uma forma organizada de apresentar que esta variável está representando uma constante, ou seja, o seu valor durante o código não deverá ser mudado. Apesar de que podemos alterar o valor desta variável a qualquer momento, pois não estamos lidando com uma tupla, por questões de organização do código, não mexeremos no valor desta variável.

Além de que, o valor 330 está um pouco estranho para a gravidade, você não acha? Vamos supor que estamos em um planeta diferente da Terra e que este possui este valor um pouco alto de gravidade. (Posteriormente você pode brincar com o código e achar uma forma dele funcionar com a gravidade da Terra)

Precisamos, agora, encontrar uma forma de calcular a velocidade vertical do pássaro. A partir da coordenada y inicial do pássaro, ele terá uma velocidade associada. Como você pensa que podemos fazer isso?

```
def obter_velocidade_y_lancamento(coordenada_y):
    if(220 >= coordenada_y < 240):
        return -200
    elif(240>= coordenada_y < 260):
        return -250
    elif(260>= coordenada_y < 280):
        return -300
    elif(280>= coordenada_y < 300):
        return -400
    elif(300>= coordenada_y < 320):
        return -450
    elif(293>= coordenada_y < 310):
        return  -500
    elif(310 >= coordenada_y < 325):
        return  -620
    else:
        return  -670
```

Se você pensou em algo assim, você está no caminho certo. Temos a função `obter_velocidade_y_lancamento`. Ela recebe como parâmetro a coordenada y.
Dentro desta função, temos que a depender desta coordenada y do pássaro, teremos como resultado a sua velocidade equivalente. Podemos fazer isso pois a velocidade de um objeto é diretamente proporcional a sua distância.

Perceba que as coordenadas que estamos utilizando dentro das comparações estão dentro do limite em que o pássaro pode se movimentar. Sendo assim, quanto menor for a movimentação dele, menor será a sua velocidade (perceba que os valores estão negativos).

Sabemos que não existe velocidade negativa, porém tivemos que colocar os valores negativos pelo fato de como cresce a coordenada y de um objeto dentro do jogo. Lembre-se da imagem que vimos na aula anterior apresentando as coordenadas do jogo.

Iremos também criar uma nova variável que irá armazenar o tempo que se passou desde o lançamento do pássaro:

```
tempo_jogo = 0
```

Inicialmente ele irá começar com 0, pois o pássaro ainda não foi lançado.

Feito isso, a função que criamos irá ficar desta maneira:

```
def movimentar_passaro(posicao_x,posicao_y):
    passaro.y   = posicao_y + (obter_velocidade_y_lancamento(posicao_y))*tempo_jogo +
(GRAVIDADE*(tempo_jogo**2))/2
```

Chamando esta função no **game loop**:

```
while(True):
    if(mouse.is_button_pressed(1)):
        coordenada_x_mouse = mouse.get_position()[0]
        coordenada_y_mouse = mouse.get_position()[1]
        if(passaro_pode_mover(coordenada_x_mouse, coordenada_y_mouse)):
            passaro.set_position(coordenada_x_mouse, coordenada_y_mouse)
            posicao_lancamento_x = coordenada_x_mouse
            posicao_lancamento_y = coordenada_y_mouse
            movimentar_passaro(posicao_lancamento_x,posicao_lancamento_y)
    tempo_jogo+=1*janela.delta_time()
```

Se você rodar o jogo, você verificará que se você clicar com o botão esquerdo do mouse, o pássaro irá subir e depois de um tempo irá descer.
Sendo assim, o nosso pássaro está **quase** pronto para se movimentar.

Vamos testar o que acabamos de fazer seguindo os seguintes requisitos:

- ➔ Se o botão esquerdo do mouse estiver pressionado: iremos capturar as coordenadas do mouse, modificar a posição do pássaro (caso ele esteja nos limites pré-estabelecidos).

➔ Caso o usuário não esteja mais pressionando o botão esquerdo, iremos chamar a função `movimentar_passaro()` para que o pássaro se movimente.

O primeiro requisito já foi feito, agora, como você poderia fazer o segundo?
Se você pensou em algo assim:

```
passaro_lancado = False
botao_clicado = False

[...]

while(True):
    if(mouse.is_button_pressed(1)):
        botao_clicado = True
        coordenada_x_mouse = mouse.get_position()[0]
        coordenada_y_mouse = mouse.get_position()[1]
              if(passaro_pode_mover(coordenada_x_mouse,  coordenada_y_mouse)  and  not
passaro_lancado):
            passaro.set_position(coordenada_x_mouse, coordenada_y_mouse)
            posicao_lancamento_x = coordenada_x_mouse
            posicao_lancamento_y = coordenada_y_mouse
    if(not mouse.is_button_pressed(1) and botao_clicado):
        botao_clicado  = False
        passaro_lancado = True
    if(passaro_lancado):
        tempo_jogo+=1*janela.delta_time()
        movimentar_passaro(posicao_lancamento_x,posicao_lancamento_y)
```

Está correto.
Inicialmente estamos criando duas variáveis, uma que verifica se o pássaro foi lançado e outra que verifica se o botão foi clicado.
Caso o botão for pressionado, colocamos o valor **True** à variável `botao_clicado`.

Agora, caso o botão não esteja pressionado e ele foi pressionado anteriormente (a checagem é feita com a variável `botao_clicado` que colocamos o valor como verdadeiro) colocamos o valor falso nessa variável (pois o botão não está mais pressionado) e colocamos como verdadeira, a variável `passaro_lancado` - significando que ele está pronto para voar.

Após isso, fazemos uma verificação da variável `passaro_lancado`, para que o pássaro só se movimente caso ele esteja pronto. E caso ele já tenha sido lançado, iremos aumentar o valor do `tempo_jogo` e movimentamos o pássaro.

Rode o seu código e lance o pássaro (clicando, arrastando e soltando) o seu pássaro deve subir e descer somente quando o botão esquerdo for solto.

Fazendo com que o pássaro se mova também na horizontal

Bom, o nosso pássaro agora precisa se movimentar na horizontal para que tenhamos um movimento oblíquo.
Assim como fizemos com lançamento vertical, também iremos criar uma função que calcula a velocidade da coordenada x de acordo com a posição inicial x do pássaro. Você imagina como podemos fazer isso?

```
def obter_velocidade_x_lancamento(coordenada_x):
    if(203 >= coordenada_x > 180):
        return 20
    elif(180 >= coordenada_x > 160):
        return 120
    elif(160 >= coordenada_x > 140):
        return 160
    elif(140 >= coordenada_x > 120):
        return 200
    elif(120 >= coordenada_x > 100):
        return 220
    elif(100 >= coordenada_x > 80):
        return 240
    elif(80 >= coordenada_x > 60):
        return 260
    elif(60 >= coordenada_x > 55):
        return 280
    else:
        return 300
```

Feito essa função, temos agora que chamá-la na função `movimentar_passaro` utilizando a função horária da posição do movimento horizontal:

$$\mathbf{x(t) = x_o + V_{ox}\, t}$$

```
def movimentar_passaro(posicao_x,posicao_y):
    passaro.y  = posicao_y + (obter_velocidade_y_lancamento(posicao_y))*tempo_jogo +
(GRAVIDADE*(tempo_jogo*tempo_jogo))/2

     passaro.x = posicao_x + obter_velocidade_x_lancamento(posicao_x)*tempo_jogo
```

Sendo assim, o pássaro já está se movimentando segundo o lançamento oblíquo. Porém, ele continua descendo e não para quando toca no chão. Originalmente, quando o pássaro toca no chão ele perde uma vida e volta para a sua posição inicial. Como você faria isso?

Um modo válido é mostrado a seguir:

```
def passaro_tocando_chao(coordenada_x,coordenada_y):
    if(passaro_pode_mover(coordenada_x, coordenada_y)):
        return False
    else:
        if(coordenada_y >=337):
            passaro.set_position(166,257)
            passaro_lancado = False
            return True
        else:
            return False
```

Ele recebe como parâmetro a coordenada x e y do pássaro e verifica se o pássaro está nas suas posições iniciais pois se ele estiver no estilingue, e enquanto você estica o pássaro, ele toca no chão, nós não queremos que ele perca uma vida. Sendo assim, fazemos essa verificação e retornamos **False**.
Caso contrário: o pássaro foi lançado. Nós verificamos se a sua coordenada é maior ou igual a 337 (ou seja, está ultrapassando a nossa imagem de chão). Caso seja verdade, ele irá perder uma vida, colocaremos o pássaro novamente na posição inicial e definimos que o pássaro não foi lançado para permitir que possamos o lançar novamente).

Feito isso, vamos chamar essa função dentro do nosso **game_loop**. Ela vai estar dentro da verificação que permita que chamemos a função `movimentar_passaro`:

```
   if(passaro_lancado and not passaro_tocando_chao(passaro.x,passaro.y)):
        tempo_jogo+=1*janela.delta_time()
        movimentar_passaro(posicao_lancamento_x,posicao_lancamento_y)
    if(passaro_tocando_chao(passaro.x,passaro.y)):
        vidas-=1
        passaro_lancado = False
        tempo_jogo = 0
```

Colocamos também uma verificação para caso o pássaro esteja tocando o chão. Se isso acontecer, ele irá perder uma vida. Colocaremos que o pássaro não foi lançado e zeramos a variável `tempo_jogo`.

Passo 02 - Detectando colisões com o pássaro

Você deve ter percebido que o pássaro está se movimentando, porém ainda quando ele encosta no pássaro, nada acontece. Vamos implementar essa funcionalidade, mas primeiro, com base nos seus conhecimentos anteriores, como você implementaria essa função?

Se você pensou em algo desse tipo:

```
def tocando_porco():
    for porco in porcos:
        if passaro.collided(porco):
            porcos.remove(porco)
            return True
    return False
```

Percorremos a lista que contém os porcos, e fazemos a verificação com cada um, caso ele esteja colidindo com o pássaro, iremos adicionar um ponto ao jogador e remover este porco da lista (ao fazer isso, ele irá desaparecer).
Vamos agora a chamar dentro do **game loop**:

```
if(passaro_lancado and not passaro_tocando_chao(passaro.x,passaro.y)):
    tempo_jogo+=1*janela.delta_time()
    movimentar_passaro(posicao_lancamento_x,posicao_lancamento_y)
    if(tocando_porco()):
        pontos+=1
```

Logo, caso o pássaro esteja tocando no porco, a função irá remover o porco da lista e irá acrescentar 1 a variável de pontos.

Passo 03 - Detectando colisões com as colunas

Assim como detectamos as colisões com os porcos, também temos que verificar as colisões com as plataformas. Atualmente, quando o pássaro encosta em uma coluna, ele consegue atravessar. Sabendo que na vida real isso não é possível. Sendo assim, temos que verificar se o pássaro está atingindo a plataforma para fazermos com que ele não continue se movimentando horizontalmente. Como você faria isso?

```
def verifica_se_passaro_atingiu_plataforma():
    for plataforma in plataformas:
        if(plataforma.collided(passaro)):
            return True
    return False
```

E por fim, a chamamos dentro da função `movimentar_passaro`:

```
def movimentar_passaro(posicao_x,posicao_y):
     passaro.y   = posicao_y +  (get_velocidade_y_lancamento(posicao_y))*tempo_jogo  +
(GRAVIDADE*(tempo_jogo*tempo_jogo))/2
    if(verifica_se_passaro_atingiu_plataforma()):
        passaro.move_x(-1*janela.delta_time())
    else:
        passaro.x = posicao_x + get_velocidade_x_lancamento(posicao_x)*tempo_jogo
```

Passo 04 - Construindo os outros níveis

Como vimos anteriormente, o nosso jogo irá conter 3 níveis. Você já fez o primeiro nível e a colocou dentro de uma função para podermos deixar o código organizado. Olhe as imagens que estão disponibilizadas acima. Como você posicionaria os objetos nos outros dois níveis?

```
def construir_mundo_nivel_2():
    global fundo, plataformas, vidas
    vidas+= 6
    fundo = GameImage('./assets/level02bg.png')
    plataformas = []
    plataforma2 = GameImage('./assets/platform2.png')
    plataforma2.set_position(760,55)
    plataforma1 = GameImage('./assets/platform1.png')
    plataforma1.set_position(900,145)
    plataforma3 = GameImage('./assets/platform3.png')
    plataforma3.set_position(980,175)
    plataforma4 = GameImage('./assets/platform1.png')
    plataforma4.set_position(1120,150)
    plataformas.append(plataforma1)
    plataformas.append(plataforma2)
    plataformas.append(plataforma3)
    plataformas.append(plataforma4)
    porco1 = Sprite('./assets/porco.png',3)
    porco1.set_position(788,13)
    porco1.set_total_duration(900)
    porco2 = Sprite('./assets/porco.png',3)
    porco2.set_position(905,102)
    porco2.set_total_duration(900)
    porco3 = Sprite('./assets/porco.png',3)
    porco3.set_position(1030,130)
    porco3.set_total_duration(900)
    porco4 = Sprite('./assets/porco.png',3)
    porco4.set_position(1131,109)
    porco4.set_total_duration(900)
    porcos.append(porco1)
    porcos.append(porco2)
    porcos.append(porco3)
    porcos.append(porco4)
```

```
def construir_mundo_nivel_3():
    global fundo, plataformas, vidas
    vidas+= 8
    fundo = GameImage('./assets/level03bg.png')
    plataformas = []
    plataforma2 = GameImage('./assets/platform2.png')
    plataforma2.set_position(550,55)
    plataforma1 = GameImage('./assets/platform1.png')
    plataforma1.set_position(720,145)
    plataforma3 = GameImage('./assets/platform2.png')
    plataforma3.set_position(780,55)
    plataforma4 = GameImage('./assets/platform1.png')
    plataforma4.set_position(900,145)
    plataforma5 = GameImage('./assets/platform2.png')
    plataforma5.set_position(960,55)
    plataforma6 = GameImage('./assets/platform1.png')
    plataforma6.set_position(1100,145)
    plataformas.append(plataforma1)
    plataformas.append(plataforma2)
    plataformas.append(plataforma3)
    plataformas.append(plataforma4)
    plataformas.append(plataforma5)
    plataformas.append(plataforma6)
    porco1 = Sprite('./assets/porco.png',3)
    porco1.set_position(584,11)
    porco1.set_total_duration(900)
    porco2 = Sprite('./assets/porco.png',3)
    porco2.set_position(730,105)
    porco2.set_total_duration(900)
    porco3 = Sprite('./assets/porco.png',3)
    porco3.set_position(825,7)
    porco3.set_total_duration(900)
    porco4 = Sprite('./assets/porco.png',3)
    porco4.set_position(905,105)
    porco4.set_total_duration(900)
    porco5 = Sprite('./assets/porco.png',3)
    porco5.set_position(1000,14)
    porco5.set_total_duration(900)
    porco6 = Sprite('./assets/porco.png',3)
    porco6.set_position(1120,105)
    porco6.set_total_duration(900)
    porcos.append(porco1)
    porcos.append(porco2)
    porcos.append(porco3)
    porcos.append(porco4)
    porcos.append(porco5)
    porcos.append(porco6)
```

Passo 05 - Mudando de fase

Para podermos trocar de fase, precisamos verificar se a nossa lista de porcos está vazia (significando que o pássaro colidiu com todos os porcos). Para uma melhor organização, iremos criar uma função que a depender do nível atual do usuário, ele irá chamar uma das funções necessárias: `construir_mundo_nivel_1()`, `construir_mundo_nivel_2()` ou `construir_mundo_nivel_3()`.

Como você poderia fazer isso?

Iniciaremos com a criação de duas variáveis. A primeira irá nos informar se podemos ir para o próximo nível e a segunda nos informará o nível atual em que o jogo se encontra.

```
proximo_nivel = False
nivel_atual = 1
```

Em seguida, criamos a seguinte função:

```
def obter_proximo_nivel():
    global proximo_nivel
    proximo_nivel = False
    if(nivel_atual == 1):
        passaro.set_position(166,257)
        construir_mundo_nivel_1()
    elif(nivel_atual ==2):
        passaro.set_position(166,257)
        construir_mundo_nivel_2()
    elif(nivel_atual ==3):
        passaro.set_position(166,257)
        construir_mundo_nivel_3()
```

E dentro do nosso **game loop**, teremos:

```
if(len(porcos) == 0):
        proximo_nivel = True
if(proximo_nivel):
    nivel_atual+=1
    obter_proximo_nivel()
```

Passo 06 - Criando uma tela de fim de jogo

Ao contrário dos jogos anteriores, nos quais quando o jogador, o jogo automaticamente se fechava. Neste iremos dar uma opção ao usuário, se ele deseja continuar ou sair do jogo:

A tela de fim de jogo você já sabe como fazer, estou certo? Tente replicar o que aparece na tela. (A imagem branca que irá aparecer está dentro da pasta assets e com o nome quadrado.png). Fazendo isso, vamos ter algo mais ou menos assim:

```
def finalizar_jogo():
    if vidas == 0 or nivel_atual>3:
        quadrado = GameImage('./assets/quadrado.png')
        quadrado.set_position(343,76)
        quadrado.draw()
                janela.draw_text("Fim  do  jogo",450,141,50,  color=(0,0,0),
font_name="Arial",bold=True, italic=False)
          janela.draw_text("Pontos: "+str(pontos),540,250,20, color=(0,0,0),
font_name="Arial",bold=True, italic=False)
            janela.draw_text("Jogar Novamente ",400,335,16, color=(0,0,0),
font_name="Arial",bold=True, italic=False)
                     janela.draw_text("Sair   ",733,335,16,   color=(0,0,0),
font_name="Arial",bold=True, italic=False)
```

Falta agora verificarmos se o usuário gostaria de sair ou jogar novamente. Como você acha que poderíamos fazer isso? (Relembre o que a gente já fez até agora...)

Se você pensou em algo que envolva as coordenadas do mouse e criação de uma função, você está correto!

Primeiramente iremos definir uma área que poderá ser “clicável” pelo jogador. Ou seja, imaginamos um retângulo ao redor de “Jogar Novamente” e “Sair”. Para cada retângulo, teremos um conjunto de coordenadas x e y onde o jogador pode clicar. Se o mouse estiver entre os valores possíveis, executaremos uma ação que poderá ser de jogar novamente ou sair. Veja:

```
def jogar_novamente():
    global vidas, passaro_lancado,pontos, porcos
    coordenada_x_mouse = mouse.get_position()[0]
    coordenada_y_mouse = mouse.get_position()[1]
     if(mouse.is_button_pressed(1)  and  402<=coordenada_x_mouse<=  535  and
329<= coordenada_y_mouse <=350):
        porcos = []
        vidas = 3
        pontos = 0
        passaro_lancado =  False
        janela.delay(2000)
        get_proximo_nivel()
```

Verificamos se o botão esquerdo está pressionado e se a coordenada x e y está dentre dois valores previamente definidos. Esses valores correspondem às "pontas" do retângulo.
Caso seja verdade essa condição, nós iremos resetar o jogo.

A mesma coisa fazemos com a função sair:

```
def sair():
    coordenada_x_mouse = mouse.get_position()[0]
    coordenada_y_mouse = mouse.get_position()[1]
     if(mouse.is_button_pressed(1)  and  732<=coordenada_x_mouse<=  764  and
329<= coordenada_y_mouse <=350):
        janela.close()
```

E por fim, a função `finalizar_jogo` ficará dessa forma:

```
def finalizar_jogo():
    if vidas == 0 or nivel_atual>3:
        quadrado = GameImage('./assets/quadrado.png')
        quadrado.set_position(343,76)
        quadrado.draw()
                        janela.draw_text("Fim do jogo",450,141,50, color=(0,0,0),
font_name="Arial",bold=True, italic=False)
                  janela.draw_text("Pontos: "+str(pontos),540,250,20, color=(0,0,0),
font_name="Arial",bold=True, italic=False)
                    janela.draw_text("Jogar Novamente ",400,335,16, color=(0,0,0),
font_name="Arial",bold=True, italic=False)
                               janela.draw_text("Sair ",733,335,16, color=(0,0,0),
font_name="Arial",bold=True, italic=False)
        jogar_novamente()
        sair()
```

E poderemos chamá-la no **game loop:**

```
janela.draw_text("Vidas:"+str(vidas),1017,12,color=(255,255,255),
font_name="Arial",bold=True, italic=False)
janela.draw_text("Pontos:"+str(pontos),50,12,color=(255,255,255),
font_name="Arial",bold=True, italic=False)

finalizar_jogo()
```

Passo 07 - Toques finais

Vamos dar alguns toques finais no nosso jogo, faremos algumas coisas que são básicas. Atualmente, antes do game loop estamos chamando diretamente a função `construir_mundo_nivel_1()`. Vamos substituir a chamada para essa função, para a chamada da função `obter_proximo_nivel()` só para termos uma melhor organização.

E vamos também, como no último jogo, definir o número de frames por segundo que esse jogo irá rodar.

Vamos ter a seguinte importação:

```
from pygame.time import *
```

E por fim, teremos o seguinte:

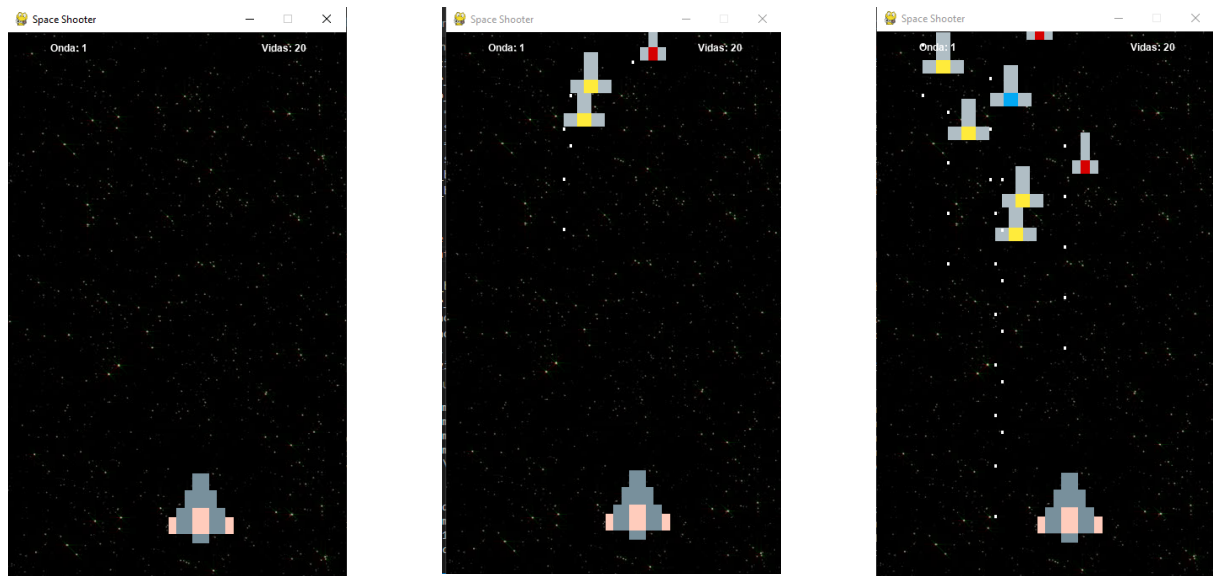
```
clock = Clock()
while(True):
    clock.tick(60)
    [...]
```

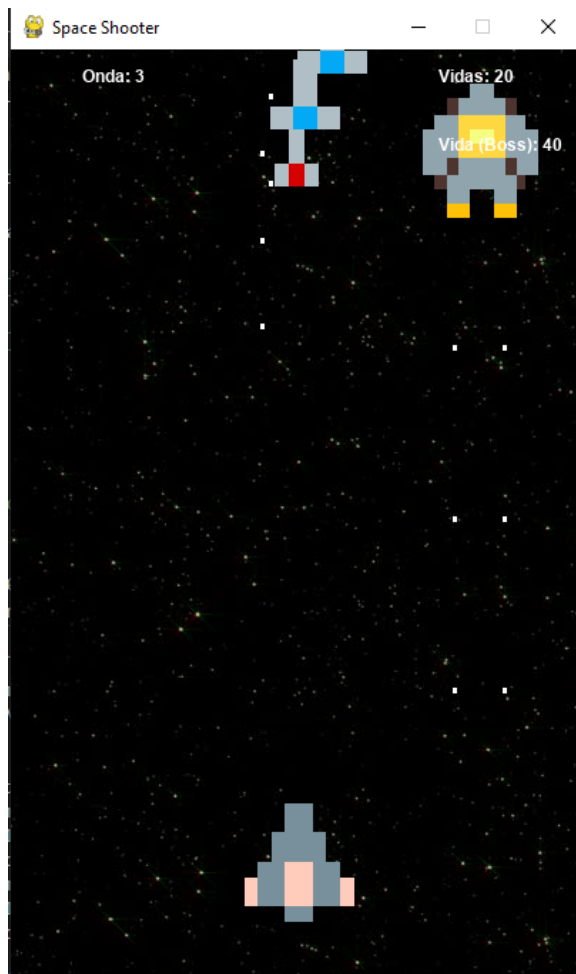
AULA 7 - Criando o jogo Space Shooter

1. Sumário

Nesta aula iremos criar o jogo Space Shooter.

FOLHA DE ATIVIDADES - Criando o jogo Space Shooter

Chegamos no nosso último jogo: O Space Shooter. Ele é inspirado em um jogo clássico conhecido como Space Invaders.

Neste jogo, poderemos controlar a nossa nave, que pode se movimentar por todas as direções. Enquanto a nave se movimenta (utilizando as setas do teclado), podemos também utilizar a tecla espaço para atirar. Enquanto a nossa nave está se movimentando e atirando, naves inimigas aparecem na tela atirando no jogador (caso a bala atinja a nossa nave, iremos perder 1 vida).

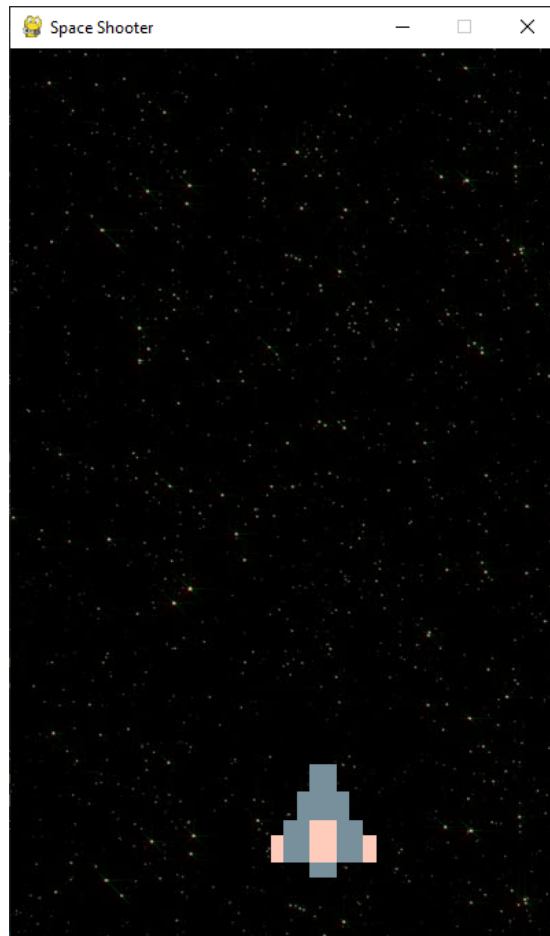
Enquanto isso, o nosso jogo terá 3 fases que estamos chamando de ondas.
Cada onda terá um número diferente de inimigos que vão aparecer. Na primeira, teremos 10 inimigos, na segunda 20 e na terceira 30. Juntamente na terceira onda, teremos a presença do chefão. Que além de ter uma quantidade de vida maior, tem a possibilidade de atirar 2 balas por vez e cada bala dá um dano de 2 pontos:

Baixe os arquivos necessários para o projeto a partir deste link:
http://bit.ly/ArquivoSpaceShooter

Passo 01 - Montando a estrutura básica do jogo

Vamos criar a tela do jogo. As dimensões da nossa tela vão ser de 400 px X 650 px. Vamos neste primeiro momento criar a tela do jogo, adicionar no fundo da tela a imagem estrelada. Com o pensamento nos jogos anteriores, como você faria?

```
from PPlay.window import *
from PPlay.sprite import *
from PPlay.keyboard import *
from PPlay.gameimage import *

janela = Window(400,650)
mouse = Window.get_mouse()
teclado = Window.get_keyboard()
janela.set_title("Space Shooter")

nave = Sprite('./assets/ship.png')
nave.set_position(181,515)

fundo = GameImage('./assets/sky.png')

while(True):
    fundo.draw()
    nave.draw()
    janela.update()
```

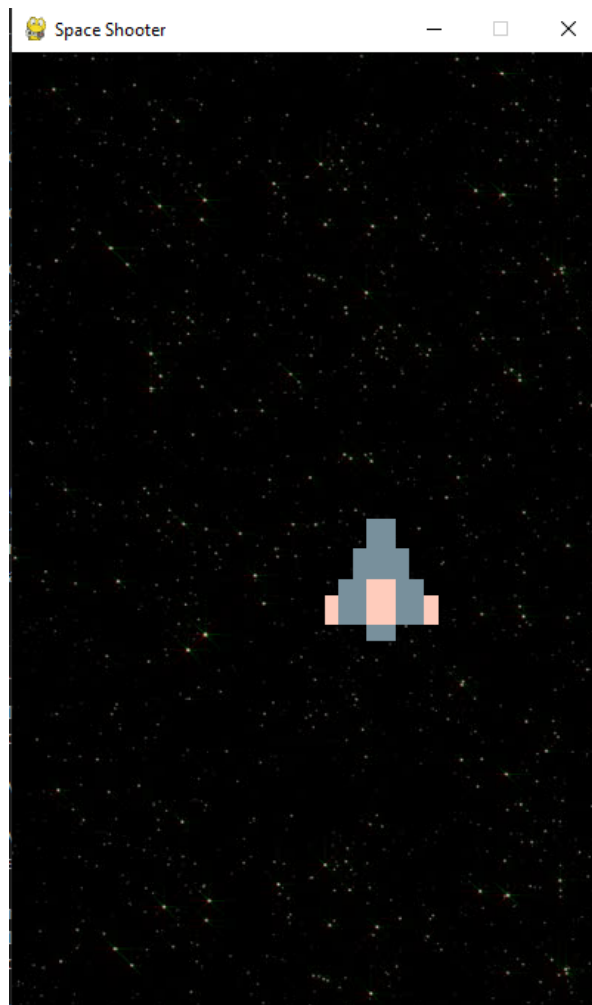
Feito isso, a seguinte tela deve aparecer:

Space Shooter

Passo 02 - Movimentando a nave

Vamos movimentar a nave pelo espaço utilizando as teclas do teclado. A partir dos seus conhecimentos prévios, tente realizar essa ação e depois olhe uma possível resposta na página seguinte.

```
def movimentar_nave(nave):
    if teclado.key_pressed("UP"):
        nave.move_y(-70*janela.delta_time())
    if(teclado.key_pressed("LEFT")):
        nave.move_x(-70*janela.delta_time())
    if(teclado.key_pressed("DOWN")):
        nave.move_y(70*janela.delta_time())
    if(teclado.key_pressed("RIGHT")):
        nave.move_x(70*janela.delta_time())
```

Lembre-se de chamar essa função dentro do nosso **game loop** e colocar o objeto de nave que criamos antes como parâmetro.
Feito isso, a nave pode se movimentar a depender da tecla pressionada:

Space Shooter

Passo 03 - Atirando

Até o momento a nave está se movimentando e não atirando. Vamos fazer com que ela possa atirar quando o usuário clicar a tecla espaço.

Para isso vamos criar uma lista que possa armazenar todos os tiros que foram lançados pela nossa nave:

```
tiros_nave = []
```

Feito isso, temos que criar uma nova função. Esta irá criar um objeto do tipo **Sprite** que irá ter a imagem da bala (a qual está dentro da pasta assets e com o nome de shoot.png).

Lembre-se que você já fez algo parecido antes, como você faria neste momento?

```
def lancar_tiro_nave():
    tiro = Sprite('./assets/shoot.png')
    tiro.set_position(nave.x+40,nave.y-5)
    tiros_nave.append(tiro)
```

Neste caso, estamos modificando a posição do tiro para que ela apareça na frente da nossa nave (por isso pegamos a coordenada x e adicionamos 40 e retiramos 5 da coordenada y).

Vamos chamar essa função que acabamos de criar, dentro da função `movimentar_nave`:

```
def movimentar_nave(nave):
    if teclado.key_pressed("UP"):
        nave.move_y(-70*janela.delta_time())
    if(teclado.key_pressed("LEFT")):
        nave.move_x(-70*janela.delta_time())
    if(teclado.key_pressed("DOWN")):
        nave.move_y(70*janela.delta_time())
    if(teclado.key_pressed("RIGHT")):
        nave.move_x(70*janela.delta_time())
    if(teclado.key_pressed("SPACE")):
        lancar_tiro_nave()
```

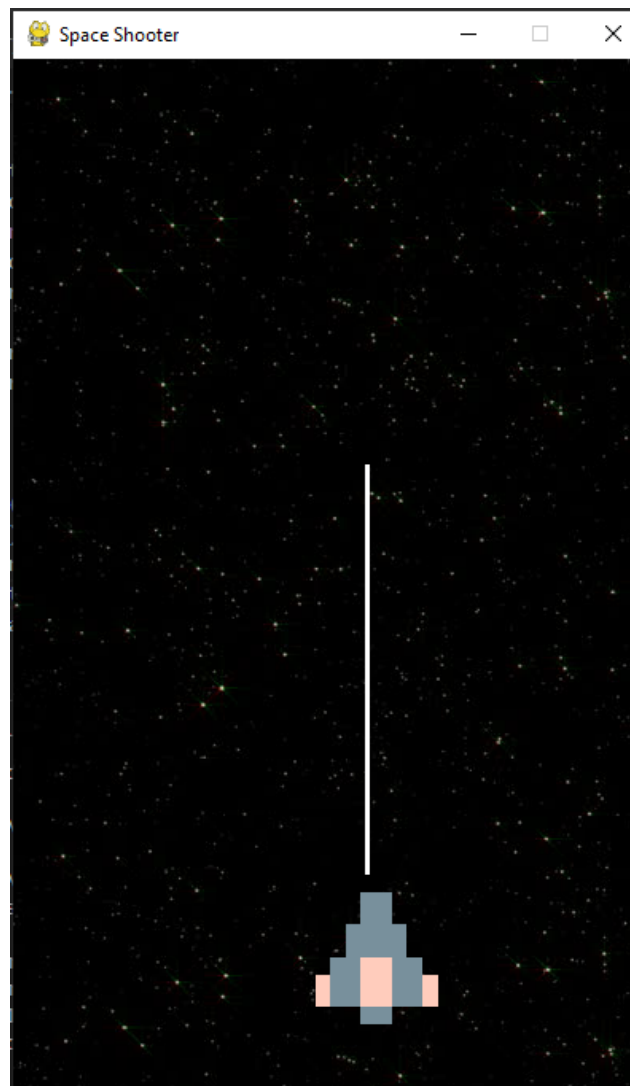
E agora, iremos fazer uma função que permita com que a bala se movimente pela tela (iremos percorrer a lista que criamos anteriormente e fazer com que a bala possa se movimentar verticalmente e caso a bala toque no limite superior, ela irá desaparecer):

```
def movimentar_tiro_nave():
    global tiros_nave
    for tiro in tiros_nave:
        if(tiro.y < 0):
            tiros_nave.remove(tiro)
        else:
            tiro.move_y(-50*janela.delta_time())
            tiro.draw()
```

Feito isso, vamos chamar essa função dentro do **game loop**:

```
while True:
    fundo.draw()
    movimentar_nave(nave)
    movimentar_tiro_nave()
    nave.draw()
    janela.update()
```

Ao fazer isso, você vai ter algo assim:

Space Shooter

Passo 04 - Arrumando a forma de atirar

Podemos ver, na imagem anterior, que os tiros estão saindo de uma só vez. Isso faz com que eles fiquem juntos formando uma linha grande. Não queremos isso, queremos que tenha um intervalo entre um tiro e outro.
Para isso, vamos primeiramente fazer a seguinte importação:

```
from pygame.time import *
```

Ao realizar essa importação, podemos fazer uso de uma função que irá nos ajudar a controlar o tempo. Podemos fazer:

```
tempo_atual = pygame.time.get_ticks()
```

O método `.time.get_ticks()` irá retornar o tempo em milissegundos que se passou desde que o jogo foi iniciado. Podemos armazenar esse valor dentro de uma variável (`tempo_atual`) e ela será útil para próximas verificações.

Vamos criar duas variáveis:

```
TEMPO_ENTRE_TIROS = 500
ultimo_tiro = 0
```

A primeira variável é uma constante que define o tempo entre dois tiros consecutivos. Associamos o valor 500 (o qual está em milissegundos).
A segunda é um valor no qual iremos armazenar quando foi a última vez em que a nave atirou.
Vamos algumas modificações nas funções que criamos:
Dentro da função `movimentar_nave`, quando a pessoa criar a tecla de espaço, primeiramente iremos pegar o tempo atual (usando o método get_ticks()) e iremos verificar: caso a subtração entre o tempo atual e a variável que armazena a última vez que a nave atirou for maior que o `TEMPO_ENTRE_TIROS`, iremos chamar a função `lancar_tiro_nave()`

```
def movimentar_nave(nave):
    if teclado.key_pressed("UP"):
        nave.move_y(-70*janela.delta_time())
    if(teclado.key_pressed("LEFT")):
        nave.move_x(-70*janela.delta_time())
    if(teclado.key_pressed("DOWN")):
        nave.move_y(70*janela.delta_time())
    if(teclado.key_pressed("RIGHT")):
        nave.move_x(70*janela.delta_time())
    if(teclado.key_pressed("SPACE")):
        tempo_atual = pygame.time.get_ticks()
        if( (tempo_atual - ultimo_tiro) > TEMPO_ENTRE_TIROS):
            lancar_tiro_nave()
```

Agora, dentro da função `lancar_tiro_nave()`, iremos adicionar um novo valor à variável tempo_atual. Esse novo valor será o retorno que obtemos a partir do `pygame.time.get_ticks()`.

```
def lancar_tiro_nave():
    global ultimo_tiro
    ultimo_tiro = pygame.time.get_ticks()
    tiro = Sprite('./assets/shoot.png')
    tiro.set_position(nave.x+40,nave.y-5)
    tiros_nave.append(tiro)
```

Sendo assim, teremos o seguinte:

Space Shooter

Passo 05 - Colocando inimigos

Vamos aos seguintes requisitos que são necessários para a criação de um inimigo na tela:

- ➔ Um inimigo irá aparecer a cada 2 segundos e o total de inimigos por fase será único, ou seja, na fase 1 só irão aparecer 10 inimigos, na segunda fase 20 e na terceira 30.
- ➔ Temos três imagens diferentes para inimigos (enemy1.png, enemy2.png e enemy3.png). Iremos escolher aleatoriamente entre as três qual é a imagem que irá aparecer do inimigo.
- ➔ Iremos fazer com que o inimigo apareça em um lugar aleatório no topo da tela
- ➔ Iremos criar uma lista que irá conter: o objeto do inimigo, a última vez em que ele atirou e uma lista contendo todos os seus tiros já lançados. E essa lista que acabamos de criar será armazenada em uma lista que conterá todos os inimigos.

A partir desses requisitos você consegue criar a função criar_inimigo() ?

```
def criar_inimigo(tempo):
```

Além disso, iremos criar algumas variáveis no início do nosso código:

```
tempo_inimigos = pygame.time.get_ticks()
nivel = 1
inimigos = []
```

Primeiramente vamos criar três variáveis, a primeira, `tempo_inimigos`, irá servir para fazer com que tenha um tempo entre o aparecimento de dois inimigos.
A segunda irá armazenar em qual nível o jogador está. Já a terceira, irá armazenar todos os objetos inimigos.

Em seguida, faremos a função:

```python
def criar_inimigo(tempo):
    global tempo_inimigos
    tempo_agora =  pygame.time.get_ticks()
    if((tempo_agora - tempo) >= 2000 and total_inimigos[nivel-1]>0):
        tempo_inimigos = pygame.time.get_ticks()
        total_inimigos[nivel-1]-=1
        aleatorio = random.randint(1,3)
        if(aleatorio == 1):
            inimigo = Sprite("./assets/enemy1.png")
        elif(aleatorio ==2):
            inimigo = Sprite('./assets/enemy2.png')
        elif(aleatorio == 3):
            inimigo = Sprite('./assets/enemy3.png')
        inimigo.set_position(random.randint(45,283), -50)
        inimigos.append([inimigo,0,[]])
```

Verificamos caso o tempo entre a última vez em que um inimigo apareceu for maior que 2000 milissegundos (2 segundos) e ainda existem inimigos naquele nível - verificamos a lista que armazena a quantidade de inimigos por nível (lembre-se que cada posição da lista contém um valor diferente). A lista que estamos falando é a seguinte:

```python
total_inimigos = [10,20,30]
```

A primeira posição (index 0) representa o primeiro nível, e por aí vai. Apesar de estarmos no primeiro nível (ou seja, o valor da variável nível = 1), se a gente fizesse `total_inimigos[nivel]` estaremos acessando a segunda posição do vetor, quando queremos na verdade acessar a primeira. Sendo assim, estamos sempre subtraindo 1 da variável nível para obter o valor correto.

Caso a condição seja verdadeira, iremos diminuir 1 na quantidade de inimigos disponíveis naquele nível (pois estamos criando um novo inimigo agora). E sorteamos um número aleatório (lembre-se de importar o **random**) para escolher qual vai ser o tipo do inimigo.
Por fim, fazemos com que o inimigo apareça em uma posição aleatória na coordenada x e colocamos esse inimigo na lista que irá armazenar todos os objetos dos inimigos.
Essa lista pode parecer um pouco complicada, mas se você observar, ela é até simples:

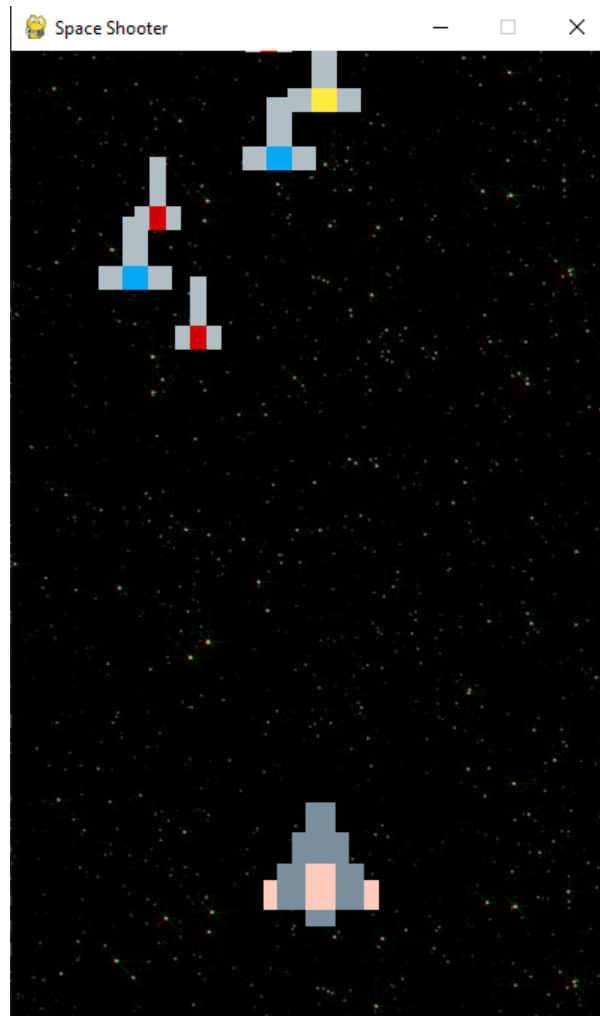
- A primeira posição da lista irá armazenar o objeto inimigo, com ele você pode fazer tudo que um objeto do tipo Sprite pode fazer
- Na segunda posição da lista, teremos a última vez em que aquele inimigo atirou, para evitarmos o problema que tivemos a uns passos atrás.
- Na terceira posição da lista, teremos também uma lista que irá armazenar todos os tiros lançados pelo inimigo.

Ao fazer isso, nada aparece na tela, pois temos que fazer uma função que irá movimentar e desenhar o inimigo na tela:

```
def mostrar_inimigos():
    for inimigo in inimigos:
        inimigo[0].move_y(20*janela.delta_time())
        inimigo[0].draw()
```

E por fim, vamos chamar as funções dentro do **game loop**:

```
while True:
    [...]
    criar_inimigo(tempo_inimigos)
    mostrar_inimigos()
    [...]
```

Ao fazer isso, teremos as naves aparecendo:

Space Shooter

AULA 8 - Concluindo o jogo Space Shooter

1. Sumário

Nesta aula iremos continuar a criação do jogo Space Shooter que iniciamos na aula anterior.

FOLHA DE ATIVIDADES - Concluindo o jogo Space Shooter

Passo 01 - Fazendo os inimigos atirarem

Até agora os inimigos estão aparecendo na tela, mas não atiram na nave. Você já fez anteriormente a nossa nave atirar, como você faria com o inimigo? Lembre-se, ao contrário da nossa nave, o inimigo vai atirar sozinho.

Podemos fazer algo assim:

Criamos uma variável que irá definir o tempo entre os tiros do inimigo.

```
tempo_entre_tiros_inimigo = 2000

def lancar_tiro_inimigo(inimigo):
    inimigo[1] = pygame.time.get_ticks()
    tiro = Sprite('./assets/shoot.png')
    tiro.set_position(inimigo[0].x,inimigo[0].y)
    inimigo[2].append(tiro)
```

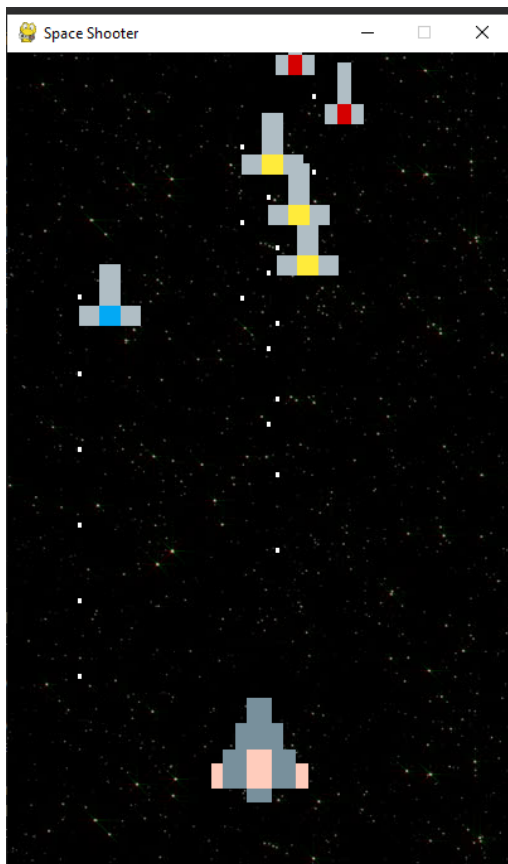
Criamos uma função que irá receber como parâmetro a lista que contém na posição 0 o objeto Sprite do inimigo, na posição 1 a última vez em que ele atirou e na última posição, a lista de tiros do inimigo.

E faremos uma modificação na função `mostrar_inimigos()`:

```
def mostrar_inimigos():
    for inimigo in inimigos:
        inimigo[0].move_y(20*janela.delta_time())
        tempo_atual = pygame.time.get_ticks()
        if( (tempo_atual - inimigo[1]) > tempo_entre_tiros_inimigo):
            lancar_tiro_inimigo(inimigo)
        inimigo[0].draw()
```

E por fim, vamos fazer a bala do inimigo se movimentar.

```python
def movimentar_tiro_inimigo():
    for inimigo in inimigos:
        for tiro in inimigo[2]:
            if(tiro.y> 650):
                inimigo[2].remove(tiro)
            else:
                tiro.move_y(50*janela.delta_time())
                tiro.draw()
```

Chamaremos a função `movimentar_tiro_inimigo()` dentro do **game loop** e teremos:

Passo 02 - Detectando a colisão entre o tiro da nossa nave e o inimigo

No jogo original, caso a bala da nossa nave colida com o inimigo, ele irá desaparecer (você fez algo desse tipo com o Angry Birds, se lembra?). Tente fazer essa implementação.

```
def mostrar_inimigos():
    for inimigo in inimigos:
        if not inimigo_atingido(inimigo):
            inimigo[0].move_y(20*janela.delta_time())
            tempo_atual = pygame.time.get_ticks()
            if( (tempo_atual - inimigo[1]) > tempo_entre_tiros_inimigo):
                lancar_tiro_inimigo(inimigo)
            inimigo[0].draw()
        else:
            inimigos.remove(inimigo)
```

Primeiro vamos fazer uma modificação na função `mostrar_inimigos`. Primeiramente percorremos a lista que armazena as informações dos inimigos. Também, iremos criar uma função, que é apresentada em baixo.

```
def inimigo_atingido(inimigo):
    for tiro in tiros_nave:
        if tiro.collided(inimigo[0]):
            return True
    return False
```

Ela irá fazer uma verificação entre as balas que estão armazenadas na lista `tiros_nave` com o inimigo que foi passado - caso elas estejam se colidindo, a função retorna **True**.

Agora voltamos para a função `mostrar_inimigos`: caso o retorno da função `inimigo_atingido` seja False, ou seja, a nossa bala não colidiu com o inimigo, significa que é para ele se movimentar. Mas caso contrário, o removemos da lista de inimigos.

Passo 03 - Detectando a colisão entre o tiro do inimigo com a nossa nave

Agora, caso o inimigo atire na nossa nave, perderemos uma vida. Essa implementação é fácil, você concorda?

```
vidas = 20
```

Primeiramente criamos uma variável que irá armazenar a quantidade de vidas disponíveis. Inicialmente o usuário terá 20 vidas.

```
def movimentar_tiro_inimigo():
    global vidas
    for inimigo in inimigos:
        for tiro in inimigo[2]:
            if(tiro.y> 650):
                inimigo[2].remove(tiro)
            else:
                tiro.move_y(50*janela.delta_time())
                tiro.draw()
            if(tiro.collided(nave)):
                inimigo[2].remove(tiro)
                vidas-=1
```

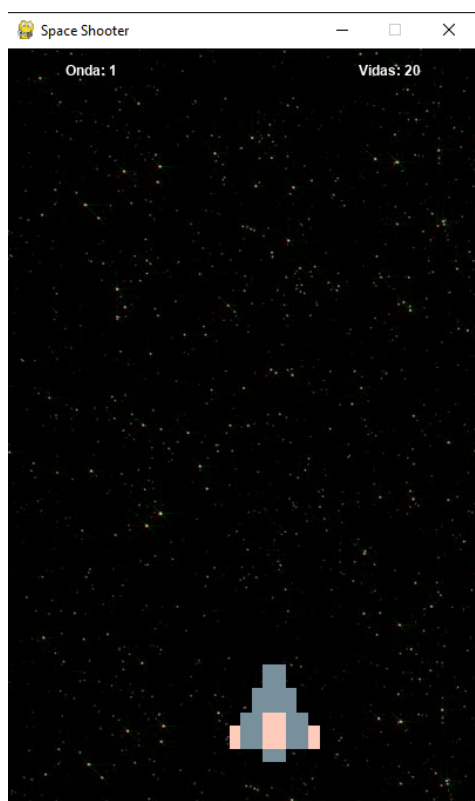
E fazemos uma pequena modificação na função `movimentar_tiro_inimigo()`.

Passo 04 - Mostrando na tela o nível atual e a quantidade de vidas disponíveis

Temos duas variáveis: nível e vidas. Precisamos mostrá-las na tela para o usuário se situar no jogo.

Dentro do **game loop**, você pode fazer algo que já fez nos jogos anteriores:

```
  janela.draw_text("Vidas: "+str(vidas),300,12, color=(255,255,255),
font_name="Arial",bold=True, italic=False)
    janela.draw_text("Onda: "+str(nivel),50,12, color=(255,255,255),
font_name="Arial",bold=True, italic=False)
```

E teremos:

Passo 05 - Indo para o próximo nível

Vamos criar uma função, assim como fizemos no Angry Birds, que verifica se o jogo pode ir para o próximo nível. No nosso caso, o próximo nível significa uma nova onda de inimigos. Como você faria isso?

```
def proxima_onda():
    global nivel
    if total_inimigos[nivel-1] == 0 and nivel <3:
        nivel+=1
```

Não esqueça de chamá-la dentro do **game loop**.

Passo 06 - Fazendo aparecer o chefão

Estamos quase finalizando o nosso jogo que até agora está muito fácil. Precisamos de um chefão que irá poder atirar duas balas por vez e cada bala contém um dano de 2 pontos.
Lembrando que, esse chefão só irá aparecer na terceira fase.

Vamos adicionar as seguintes variáveis no início do nosso código:

```
boss = Sprite("./assets/boss.png")
mostrar_boss = False
vida_boss = 40
```

Iremos criar um objeto do tipo Sprite que terá a imagem do chefão (armazenado dentro da pasta *assets* e com o nome de *boss.png*).
Inicialmente ele não será mostrado no jogo, sendo assim, criamos uma variável `mostrar_boss` que irá começar com o valor **False**. Por fim, temos uma variável que irá armazenar a vida do boss (isso mesmo, ele não irá desaparecer tão rapidamente com somente um tiro).

Para fazer com que ele apareça é bem simples, basta fazermos uma pequena modificação na função `criar_inimigo`:

```
def criar_inimigo(tempo):
    global tempo_inimigos, mostrar_boss
    tempo_agora =  pygame.time.get_ticks()
    if((tempo_agora - tempo) >= 2000 and total_inimigos[nivel-1]>0):
        tempo_inimigos = pygame.time.get_ticks()
        total_inimigos[nivel-1]-=1
        aleatorio = random.randint(1,3)
        if(aleatorio == 1):
            inimigo = Sprite("./assets/enemy1.png")
        elif(aleatorio ==2):
            inimigo = Sprite('./assets/enemy2.png')
        elif(aleatorio == 3):
            inimigo = Sprite('./assets/enemy3.png')
        inimigo.set_position(random.randint(45,283), -50)
        inimigos.append([inimigo,0,[]])
    if nivel == 3 and not mostrar_boss:
        mostrar_boss = True
        boss.set_position(random.randint(45,283),-50)
```

Temos que, caso o nível for igual a 3 e a variável `mostrar_boss` for falsa (o inimigo ainda não apareceu), iremos colocar essa variável igual a **True** e fazer com que ele apareça em uma posição aleatória.

Passo 07 - Movimentando o chefão

O chefão terá um movimento diferente do resto dos inimigos. Inicialmente ele irá descer normalmente e quando chegar a um certo ponto, ele só irá se movimentar horizontalmente. Consegue imaginar como podemos fazer isso?

Podemos iniciar com a declaração de algumas variáveis:

```
movimento_boss_baixo = True
movimento_boss_direita = False
```

Vamos criar inicialmente duas variáveis que irão nos ajudar a saber se o boss pode se movimentar horizontalmente ou verticalmente.

```
def boss_movimenta_vertical():
    global movimento_boss_baixo, boss
    if boss.y>=280:
        movimento_boss_baixo = False
    elif boss.y<= 120:
        movimento_boss_baixo = True
```

Criamos a função `boss_movimenta_vertical` que irá verificar se o chefão pode se movimentar verticalmente, ou seja: inicialmente ele irá descer como outro vilão e quando chegar em uma determinada coordenada y, ele não poderá mais.

Agora criaremos uma função que irá ajudar a fazer a movimentação horizontal do chefão.
Verificamos a coordenada x do chefão e se ele estiver entre 0px e 10px - significa que ele pode se movimentar para a direita. Caso contrário, se essa coordenada estiver entre 300px e 310px, significa que o chefão pode se movimentar para a esquerda.

```
def boss_movimenta_horizontal():
    global boss, movimento_boss_direita
    if 0 <= boss.x<=10:
        movimento_boss_direita = True
    elif 300<=boss.x<=310:
         movimento_boss_direita = False
```

Por fim, vamos criar a função que irá fazer com que o chefão se movimente de fato:

```python
def movimentar_boss():
    global boss,vida_boss
    if mostrar_boss:
        boss_movimenta_vertical()
        boss_movimenta_horizontal()
        if movimento_boss_baixo:
            boss.move_y(10*janela.delta_time())
        else:
            if movimento_boss_direita:
                boss.move_x(10*janela.delta_time())
            else:
                boss.move_x(-10*janela.delta_time())
        boss.draw()
```

Caso o chefão esteja aparecendo na tela, chamamos as funções que permitem alterar os valores das variáveis que que ajudam a fazer o movimento horizontal ou vertical do chefão.
Em seguida, fazemos os movimentos.

Chamamos a função `movimentar_boss()` dentro do game loop, e quando o jogo estiver na terceira onda, o boss vai estar se movimentando na tela.

Passo 08 - Fazendo com que o chefão atire

Para o chefão conseguir atirar, vamos ter a mesma lógica que fizemos anteriormente. A única diferença é que o chefão irá atirar duas balas por vez.

Vamos começar com a declaração das duas variáveis que vão ser importantes:

```
tiros_boss = []
ultimo_tiro_boss = 0
```

Criaremos a função que irá criar os objetos de tiro do chefão, lembrando que iremos criar dois objetos de vez, pois o chefão atira dois tiros por vez.

```
def lancar_tiro_boss():
    global ultimo_tiro_boss
    if mostrar_boss:
        ultimo_tiro_boss = pygame.time.get_ticks()
        tiro1 = Sprite('./assets/shoot.png')
        tiro1.set_position(boss.x+30,boss.y+100)
        tiro2 = Sprite('./assets/shoot.png')
        tiro2.set_position(boss.x+65,boss.y+100)
        tiros_boss.append(tiro1)
        tiros_boss.append(tiro2)
```

E por fim, fazemos uma modificação na função `movimentar_boss` para fazer com que o chefão possa atirar automaticamente.

```
def movimentar_boss():
    global boss,vida_boss
    if mostrar_boss:
        boss_movimenta_vertical()
        boss_movimenta_horizontal()
        if movimento_boss_baixo:
            boss.move_y(10*janela.delta_time())
        else:
            if movimento_boss_direita:
                boss.move_x(10*janela.delta_time())
            else:
                boss.move_x(-10*janela.delta_time())
        tempo_atual = pygame.time.get_ticks()
        if( (tempo_atual - ultimo_tiro_boss) > tempo_entre_tiros_inimigo):
            lancar_tiro_boss()

        boss.draw()
```

Por fim, vamos criar a função que permite com que o tiro do chefão possa aparecer na tela:

```
def movimentar_tiro_boss():
    for tiro in tiros_boss:
        tiro.move_y(70*janela.delta_time())
        tiro.draw()
```

Não esqueça de chamá-la no **game loop**

Passo 09 - Disponibilizando a vida do chefão

Agora que o chefão está aparecendo na tela, podemos disponibilizar para o jogador a quantidade de vidas que ele possui naquele momento.
No **game loop**, coloque o seguinte código:

```
    if(mostrar_boss):
        janela.draw_text("Vida (Boss): "+str(vida_boss),300,60,
color=(255,255,255), font_name="Arial",bold=True, italic=False)
```

E teremos:

Passo 10 - Lidando com a colisão

Assim como fizemos com as naves inimigas, vamos fazer também com o chefão. Caso a bala do chefão atinja a nossa nave, vamos perder dois pontos de vida e caso a bala da nossa nave atinja o chefão, ele irá perder 1 ponto de vida:

```
def movimentar_boss():
    global boss,vida_boss
    if mostrar_boss:
        boss_movimenta_vertical()
        boss_movimenta_horizontal()
        if movimento_boss_baixo:
            boss.move_y(10*janela.delta_time())
        else:
            if movimento_boss_direita:
                boss.move_x(10*janela.delta_time())
            else:
                boss.move_x(-10*janela.delta_time())
        if(tiro_atingiu()):
            vida_boss-=1
        tempo_atual = pygame.time.get_ticks()
        if( (tempo_atual - ultimo_tiro_boss) > tempo_entre_tiros_inimigo):
            lancar_tiro_boss()

        boss.draw()
```

Dentro da função `movimentar_boss`, iremos chamar a função `tiro_atingiu_boss`:

```
def tiro_atingiu_boss():

    for tiro in tiros_nave:
        if mostrar_boss:
            if tiro.collided(boss):
                tiros_nave.remove(tiro)
                return True
    return False
```

Ao fazer isso, quando a bala da nossa nave atinge o chefão, ele perde 1 ponto de vida.

Para fazer com que a nossa nave perca dois pontos de vida caso a bala do chefão a atinja, podemos fazer a seguinte modificação na função `movimentar_tiro_boss`:

```
def movimentar_tiro_boss():
    global vidas
    for tiro in tiros_boss:
        tiro.move_y(70*janela.delta_time())
        tiro.draw()
        if(tiro.collided(nave)):
            tiros_boss.remove(tiro)
            vidas-=2
```

Passo 11 - Finalizando o jogo

O nosso jogo está bem perto de acabar. Precisamos agora verificar se a quantidade de vidas do jogador é menor do que zero ou até mesmo se a vida do chefão é igual a zero.
Faremos a mesma coisa que fizemos no jogo anterior, disponibilizando opções para o jogador escolher se ele deseja finalizar o jogo ou continuar.
Como você realiza essa implementação?

Podemos fazer o seguinte:
Vamos criar uma variável que irá nos ajudar a saber se o jogo finalizou ou não. Caso ele não esteja finalizado, iremos permitir as movimentações dos objetos na tela.

```
fim_jogo = False
```

Vamos criar a função `verificar_fim_jogo()`:

```
def verificar_fim_jogo():
    global fim_jogo
    if vidas <= 0 or vida_boss <= 0:
        fim_jogo = True
        quadrado = GameImage('./assets/quadrado.png')
        quadrado.set_position(8,250)
        quadrado.draw()
        janela.draw_text("Fim do jogo",40,300,50, color=(0,0,0),
font_name="Arial",bold=True, italic=False)
        janela.draw_text("Jogar Novamente ",110,400,16,
color=(0,0,0), font_name="Arial",bold=True, italic=False)
        janela.draw_text("Sair ",160,430,16, color=(0,0,0),
font_name="Arial",bold=True, italic=False)
        jogar_novamente()
        sair()
```

A função `jogar_novamente`:

```
def jogar_novamente():
    global vidas, vida_boss, total_inimigos, tempo_inimigos, nivel,
fim_jogo, tiros_nave, tiros_inimigos, tiros_boss, inimigos,mostrar_boss
    coordenada_x_mouse = mouse.get_position()[0]
    coordenada_y_mouse = mouse.get_position()[1]
    if(mouse.is_button_pressed(1) and 108<=coordenada_x_mouse<= 244 and
395<= coordenada_y_mouse <=414):
        vidas = 20
        vida_boss = 40
        total_inimigos = [10,20,30]
        nave.set_position(181,515)
        tiros_nave = []
        tiros_inimigos = []
        tiros_boss = []
        inimigos = []
        mostrar_boss = False
        tempo_inimigos = pygame.time.get_ticks()
        nivel = 1
        fim_jogo = False
        janela.delay(2000)
```

E `sair()`:

```
def sair():
    coordenada_x_mouse = mouse.get_position()[0]
    coordenada_y_mouse = mouse.get_position()[1]
    if(mouse.is_button_pressed(1) and 158<=coordenada_x_mouse<= 188 and
425<= coordenada_y_mouse <=440):
        janela.close()
```

Por fim, o **game loop** ficará assim:

```
while(True):
    fundo.draw()
    if not fim_jogo:
        movimentar_nave(nave)
        nave.draw()
        mostrar_inimigos()
        movimentar_tiro_nave()
        movimentar_tiro_inimigo()
        movimentar_tiro_boss()
        criar_inimigo(tempo_inimigos)
        proxima_onda()
        movimentar_boss()
    verificar_fim_jogo()
    janela.draw_text("Vidas: "+str(vidas),300,12, color=(255,255,255),
font_name="Arial",bold=True, italic=False)
    janela.draw_text("Onda: "+str(nivel),50,12, color=(255,255,255),
font_name="Arial",bold=True, italic=False)
    if(mostrar_boss):
        janela.draw_text("Vida (Boss): "+str(vida_boss),300,60,
color=(255,255,255), font_name="Arial",bold=True, italic=False)

    janela.update()
```

E você finaliza o seu jogo! 😁

AULA 9 - Fazendo o seu jogo

1. Sumário

A partir do conhecimento que você obteve com as últimas aulas a respeito da construção de jogos digitais, agora chegou a sua vez de criar o **seu** próprio jogo. Libere a criatividade e construa um jogo totalmente do zero. Se tiver alguma dúvida a respeito de algum conteúdo, chame o seu professor ou releia as aulas passadas.

FOLHA DE ATIVIDADES

Crie um Game Design Document (GDD) para o seu jogo com as seguintes informações:

Nome do jogo:

____

Tipo do jogo (marque com o x):

[ ] Carro [ ] Plataforma [ ] Running [ ] Tiro [ ] Puzzle [ ] Estratégia

[ ] Outro: ____

Objetivo do jogo:

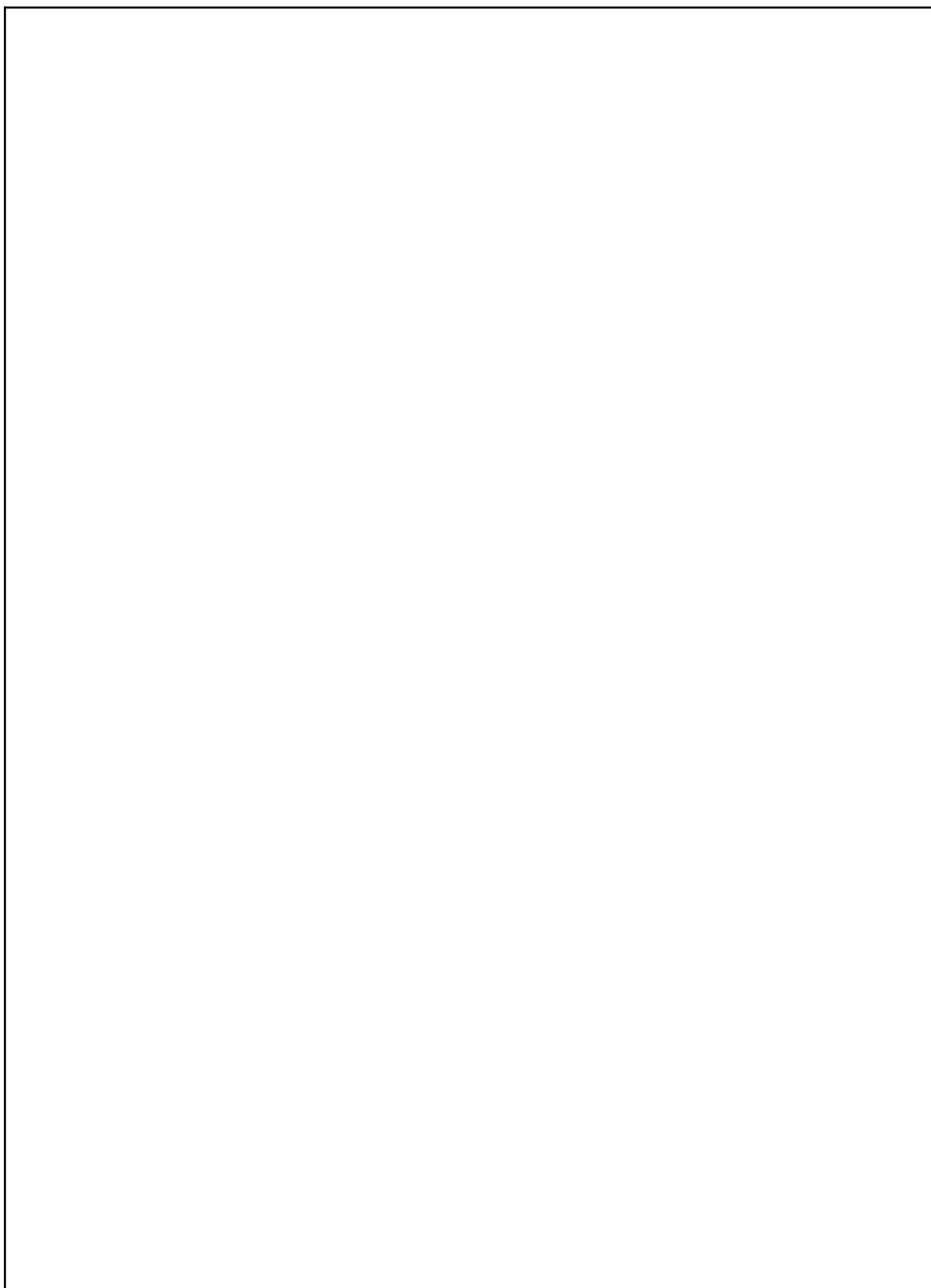
____

____

____

Funções Utilizadas (marque com o x):

[ ] Pontuação [ ] Vidas [ ] Números Aleatórios [ ] Colisão [ ] Interação com o Mouse

[ ] Interação com o Teclado [ ] Personagens não jogáveis (*Non-Player Characters - NPCs*)

[ ] Criação de objetos a partir de outros (spawn)

[ ] Outros:

____

____

Personagens/Itens

Nome	Função/Mecânica	Sprite

Regras do jogo (Descrição ou Desenhos):

AULA 10 - Finalizando o seu jogo

1. Sumário

Nesta aula você irá concluir a construção do seu jogo e apresentará para o seu professor e seus colegas.